(19) Japanese Patent Office (JP) (12) **Patent Gazette (B2)** (11) Japanese Patent No.

2966085

(45) Issue Date: October 25, 1999 (24) Registration Date: August 13, 1999

(51) Int. Cl.[6]	ID Symbol	FI	
G 06 F 9/34	340	G 06 F 9/34	340A
9/38	310	9/38	310A

Number of claims: 4 (Total of 41 pages)

(21) Application No.	H2-511130
(86) (22) Filing Date:	August 2, 1990
(65) Publication No.:	Published Japanese Translation of PCT International Patent Application Publication (*Kōhyō*) No. H5-502125
(43) Publication Date:	April 15, 1993
(86) International Application Number:	PCT/US90/04245
(87) International Publication Number:	WO 91/02311
(87) International Publication Date:	February 21, 1991
Date of Request for Examination:	August 4, 1997
(31) Priority Claim Number:	389,334
(32) Priority Date:	August 3, 1989
(33) Priority Claim Country:	United States (US)

(73) Patentee: 999999999
Moore, Charles H.
410 Star Hill Road, Woodside, CA 94062
United States of America

(73) Patentee: 999999999
Fish, Russell H., III
750 Shoreline Boulevard, Suite 85, Mountain View, CA 94062
United States of America

(74) Agent: Minoru Nakamura, patent attorney (and 6 others)

Examiner: Yuji Nakano

Continued on back page

Translation of Full Text of Japanese Patent No. 2,966,085

Japanese	English
(54)【発明の名称】後入れ先出しスタックを備えるマイクロプロセッサ、マイクロプロセッサシステム、及び後入れ先出しスタックの動作方法	(54) [Title of the Invention] Microprocessor provided with a push down stack, microprocessor system, and method for operating a push down stack
(57)【特許請求の範囲】 【請求項１】算術論理装置、前記の算術論理装置に接続された第１の後入れ先出しスタック、及びレジスタファイルからなり、前記の第１の後入れ先出しスタックが前記の算術論理装置の第１の入力に接続されたトップ項目を記憶する手段と前記の算術論理装置の第２の入力に接続されたネクスト項目を記憶する手段を含み、前記の算術論理装置が前記のトップ項目を記憶する手段に接続された出力を有し、前記のトップ項目を記憶する手段が前記のレジスタファイルに入力を提供するように接続されたマイクロプロセッサシステム。	(57) [Claims] [Claim 1] A microprocessor system, comprising an arithmetic logic unit, a first push down stack connected to said arithmetic logic unit, said first push down stack including means for storing a top item connected to a first input of said arithmetic logic unit and means for storing a next item connected to a second input of said arithmetic logic unit, said arithmetic logic unit having an output connected to said means for storing a top item, said means for storing a top item being connected to supply an input to said register file.
【請求項２】中央処理装置、メモリ、前記の中央処理装置を前記のメモリに接続するバスからなり、前記の中央処理装置は算術論理装置と前記の算術論理装置に接続された後入れ先出しスタックを含み、前記の後入れ先出しスタックが前記の算術論理装置の第１の入力に接続されたトップ項目を記憶する手段と前記の算術論理装置の第２の入力に接続されたネクスト項目を記憶する手段を含み、前記の算術論理装置が前記のトップ項目を記憶する手段に接続された出力を有し、前記の後入れ先出しスタックがラッチとして構成された第１の複数のスタック要素と、ランダムアクセスメモリとして構成された第２の複数のスタック要素とを有し、前記の第１および第２の複数のスタック要素と前記の中央処理装置が単一の集積回路内に設けられ、前記の後入れ先出しスタックがさらに前記の単一の集積回路の外部のランダムアクセスメモリとして構成された第３の複数のスタック要素を有するマイクロプロセッサシステム。	[Claim 2] A microprocessor system, comprising a central processing unit, a memory, a bus connecting said central processing unit to said memory, said central processing unit including an arithmetic logic unit and a push down stack connected to said arithmetic logic unit, said push down stack including means for storing a top item connected to a first input of said arithmetic logic unit and means for storing a next item connected to a second input of said arithmetic logic unit, said arithmetic logic unit having an output connected to said means for storing a top item, said push down stack having a first plurality of stack elements configured as latches and a second plurality of stack elements configured as a random access memory, said first and second plurality of stack elements and said central processing unit being provided in a single integrated circuit, said push down stack further having a third plurality of stack elements configured as a random access memory external to said single integrated circuit.
【請求項３】単一の集積回路中のメイン中央処理装置と別の直接メモリアクセス中央処理装置とからなるマイクロプロセッサであ	[Claim 3] A microprocessor comprising a main central processing unit and a separate direct memory access central processing unit within a single

り、前記のメイン中央処理装置は算術論理装置、前記の算術論理装置に入力を提供するように接続されたトップ項目レジスタとネクスト項目レジスタ付きの第１の後入れ先出しスタックを有し、前記の算術論理装置の出力は前記のトップ項目レジスタに接続され、前記のトップ項目レジスタはまた内部データバスに入力を提供するように接続され、前記の内部データバスはループカウンタに双方向に接続され、前記のループカウンタは減分器に接続され、前記の内部データバスはスタックポインタ、リターンスタックポインタ、モードレジスタ、および命令レジスタに双方向に接続され、前記の内部データバスはメモリコントローラ、リターン後入れ先出しスタックのＹレジスタ、Ｘレジスタおよびプログラムカウンタに接続され、前記のＹレジスタ、Ｘレジスタおよびプログラムカウンタは内部アドレスバスへの出力を提供し、前記の内部アドレスバスは前記のメモリコントローラへのと増分器への入力を提供し、前記の増分器は前記の内部データバスに接続され、前記の直接メモリアクセス中央処理装置は前記のメモリコントローラへの入力を提供し、前記のメモリコントローラはアドレス／データバスとランダムアクセスメモリへの接続のための複数の制御ラインを有するマイクロプロセッサ。	integrated circuit, said main central processing unit having an arithmetic logic unit, a first push down stack with a top item register and a next item register, connected to supply inputs to said arithmetic logic unit, an output of said arithmetic logic unit being connected to said top item register, said top item register also being connected to supply inputs to an internal data bus, said internal data bus being bidirectionally connected to a loop counter, said loop counter being connected to a decrementer, said internal data bus being bidirectionally connected to a stack pointer, return stack pointer, mode register and instruction register, said internal data bus being connected to a memory controller, to a Y register of a return push down stack, an X register and a program counter, said Y register, X register and program counter supplying outputs to an internal address bus, said internal address bus supplying inputs to said memory controller and to an incrementer, said incrementer being connected to said internal data bus, said direct memory access central processing unit supplying inputs to said memory controller, said memory controller having an address/data bus and a plurality of control lines for connection to a random access memory.
【請求項４】マイクロプロセッサシステムにおいて後入れ先出しスタックを動作させる方法であって、ラッチとして構成された第１の複数のスタック要素、ランダムアクセスメモリとして構成された第２の複数のスタック要素を提供し、第１および第２の複数のスタック要素をマイクロプロセッサとともに単一の集積回路内に設け、この単一の集積回路の外部のランダムアクセスメモリとして構成された第３の複数のスタック要素を提供し、後入れ先出しスタックに項目を記憶させ、第２の複数のスタック要素にアクセスすることなく第１の複数のスタック要素から第１の複数の項目を取り出し、第１の複数のスタック要素が空であるときは第２の複数のスタック要素から第１の複数の項目を取り出し、第３の複数のスタック要素にアクセスすることなく第２の複数のスタック要素から第２の複数の項目を取り出し、第２の複数のスタック要素が空であるときは第３の複数のスタック要素から第２の複数	[Claim 4] In a microprocessor system, a method for operating a push down stack, said method comprising supplying a first plurality of stack elements configured as latches, a second plurality of stack elements configured as a random access memory, the first and second plurality of stack elements being provided in a single integrated circuit with the processor, supplying a third plurality of stack elements configured as a random access memory external to the single integrated circuit, storing items in the push down stack, fetching a first plurality of items from the first plurality of stack elements without accessing the second plurality of stack elements, fetching a first plurality of items from the second plurality of stack elements when said first plurality of stack elements are empty, fetching the second plurality of items from the second plurality of stack elements without accessing the third plurality of stack elements, and fetching a second plurality of items from the third plurality of stack elements when said second plurality of stack elements are empty.

の項目を取り出す方法。	
【発明の詳細な説明】	[Detailed Description of the Invention]
背景技術	BACKGROUND OF THE INVENTION
1. 技術分野	1. Field of the Invention
この発明は広義には簡単な低減命令セットコンピュータ（RISC）マイクロプロセッサに関する。より詳細には、かかるマイクロプロセッサで、たとえば 20 ドルの価格でたとえば毎秒２、000 万の命令（MIPS）の性能レベルが可能なものに関する。	The present invention relates generally to a simplified, reduced instruction set computer (RISC) microprocessor. More particularly, it relates to such a microprocessor which is capable of performance levels of, for example, 20 million instructions per second (MIPS) at a price of, for example, 20 dollars.
2. 従来の技術	2. Description of the Prior Art
マイクロプロセッサの発明以来、その設計の改善には二つの異なる方法が用いられてきた、第１の方法ではマイクロプロセッサ集積回路中のトランジスタをより多くしまたより高速にし、さらにより複雑な命令セットを備えることによって性能の向上が図られた。この方法は、Motorola 68000 や Intel 80X86 のシリーズにその例を見ることができる。この方法は数百のピンアウトを有するより大きなダイおよびパッケージに向かう傾向にある。	Since the invention of the microprocessor, improvements in its design have taken two different approaches. In the first approach, a gain in performance has been achieved through the provision of greater numbers of faster transistors in the microprocessor integrated circuit and an instruction set of increased complexity. This approach is exemplified by the Motorola 68000 and Intel 80X86 microprocessor families. The trend in this approach is to larger die sizes and packages, with hundreds of pinouts.
最近では、マイクロプロセッサ集積回路自体とその命令セットの両方においてより簡単に性能の改善を得ることができると考えられている。この第２の方法は RISC マイクロプロセッサを提供するものであり、Sun SPARC や Intel のマイクロプロセッサにその例を見ることができる。しかし、この方法においても従来の実施態様では多数のピンアウトを収容するためにマイクロプロセッサのパッケージは大型となっている。したがって、高性能マイクロプロセッサの簡略化が依然として要請されている。	More recently, it has been perceived that performance gains can be achieved through comparative simplicity, both in the microprocessor integrated circuit itself and in its instruction set. This second approach provides RISC microprocessors, and is exemplified by the Sun SPARC and the Intel microprocessors. However, even with this approach as conventionally practiced, the packages for the microprocessor are large, in order to accommodate the large number of pinouts that continue to be employed. A need therefore remains for further simplification of high performance microprocessors.
従来の高性能マイクロプロセッサでは、そのマイクロプロセッサに対応するのに十分な速度のメモリアクセスを行うためにマイクロプロセッサへの直接接続を行うための高速スタティックメモリが必要とされる。低速のダイナミックランダムアクセスメモリ（DRAM）がこのようなマイクロプロセッサに用いられるのは、階層的メモリ構成においてのみであり、スタティックメモリがマイクロプロセッサと DRAM の間のバッファとしてはたらく。	With conventional high performance microprocessors, fast static memories are required for direct connection to the microprocessors in order to allow memory accesses that are fast enough to handle the microprocessors. Slow dynamic random access memories (DRAMs) are used with such microprocessors only in a hierarchical memory arrangement, with the static memories acting as a buffer between the microprocessors and the DRAMs.
従来のマイクロプロセッサは、マイクロプロセッサ集積回路上	Conventional microprocessors provide direct memory accesses

に設けるかあるいは別途設けられる DMA コントローラを介してシステム周辺装置のための直接メモリアクセス（DMA）を提供する。このような DMA コントローラは DMA 要求および DMA 応答のルーチン処理を提供することができるが、ある種の処理はマイクロプロセッサの主中央処理装置（CPU）によって行わなければならない。	(DMA) for system peripheral units through DMA controllers, which may be located on the microprocessor integrated circuit, or provided separately. Such DMA controllers can provide routine handling of DMA requests and responses, but some processing by the main central processing unit (CPU) of the microprocessor is required.
発明の開示	SUMMARY OF THE INVENTION
この発明の目的は低コストで高性能なマイクロプロセッサ及びマイクロプロセッサシステムを提供することである。	It is an object of this invention to provide a low cost, high performance microprocessor and microprocessor system.
この目的およびそれに関連する目的はここに開示する新しい高性能の低コストマイクロプロセッサを用いることによって達成することができる。	The attainment of these and related objects may be achieved through use of the novel high performance, low cost microprocessor herein disclosed.
この発明の一態様においては、マイクロプロセッサシステムは算術論理演算装置を含む。第１の後入れ先出しスタックがこの算術論理演算装置に接続されている。この第１の後入れ先出しスタックはこの算術論理演算装置の第１の入力に接続されたトップ項目を記憶する手段と、この算術論理演算装置の第２の入力に接続されたネクスト（次）項目を記憶する手段を含む。この算術論理演算装置はトップ項目を記憶する手段に接続された出力を有する。トップ項目を記憶する手段はレジスタファイルに入力を提供するように接続される。	In one aspect of the invention, a microprocessor system includes an arithmetic logic unit. A first push down stack is connected to the arithmetic logic unit. The first push down stack includes means for storing a top item connected to a first input of the arithmetic logic unit and means for storing a next item connected to a second input of the arithmetic logic unit. The arithmetic logic unit has an output connected to the means for storing a top item. The means for storing a top item is connected to provide an input to a register file.
この発明の別の態様においては、マイクロプロセッサシステムは中央処理装置、メモリ、中央処理装置をメモリに接続するバスを含む。中央処理装置は、算術論理装置と算術論理装置に接続された後入れ先入れスタックを含む。後入れ先出しスタックはこの算術論理演算装置の第１の入力に接続されたトップ項目を記憶する手段と、この算術論理演算装置の第２の入力に接続されたネクスト（次）項目を記憶する手段を含む。算術論理演算装置はトップ項目を記憶する手段に接続された出力を有する。後入れ先出しスタックはラッチとして構成された第１の複数のスタック要素と、ランダムアクセスメモリとして構成された第２複数のスタック要素を有する。第１および第２の複数のスタック要素と中央処理装置は単一の集積回路内に設けられている。そして、後入れ先	In another aspect of the invention, a microprocessor system includes a central processing unit, a memory, a bus connecting the central processing unit to the memory. The central processing unit includes an arithmetic logic unit and a push down stack connected to the arithmetic logic unit. The push down stack includes means for storing a top item connected to a first input of the arithmetic logic unit and means for storing a next item connected to a second input of the arithmetic logic unit. The arithmetic logic unit has an output connected to the means for storing a top item. The push down stack has a first plurality of stack elements configured as latches and a second plurality of stack elements configured as a random access memory. The first and second plurality of stack elements and the central processing unit are provided in a single integrated circuit. Moreover, a third plurality of stack elements is configured as a random access memory external to the single integrated

出しスタックが単一の集積回路の外部のランダムアクセスメモリとして構成される第３の複数のスタック要素を有する。	circuit.
この発明の更に別の態様においては、マイクロプロセッサは単一の集積回路中のメイン中央処理装置と別の直接メモリアクセス中央処理装置とを含む。メイン中央処理装置は算術論理装置、算術論理装置に入力を提供するように接続されたトップ項目レジスタとネクスト（次）項目レジスタ付きの第１の後入れ先出しスタックを有し、算術論理装置の出力はトップ項目レジスタに接続されており、トップ項目レジスタは内部データバスに入力を提供するように接続されており、内部データバスはループカウンタに双方向に接続されており、ループカウンタは減分器に接続されており、内部データバスはスタックポインタ、リターンスタックポインタ、モードレジスタ、および命令レジスタに双方向に接続されており、内部データバスはメモリコントローラ、リターン後入れ先出しスタックのＹレジスタ、Ｘレジスタおよびプログラムカウンタに接続されており、Ｙレジスタ、Ｘレジスタおよびプログラムカウンタは内部アドレスバスへの出力を提供し、内部アドレスバスはメモリコントローラへのと増分器への入力を提供し、増分器は内部データバスに接続され、直接メモリアクセス中央処理装置はメモリコントローラへの入力を提供し、メモリコントローラはアドレス／データバスとランダムアクセスメモリへの接続のための複数の制御ラインを有するマイクロプロセッサ。	In a further aspect of the invention, a microprocessor includes a main central processing unit and a separate direct memory access central processing unit within a single integrated circuit. The main central processing unit has an arithmetic logic unit, a first push down stack with a top item register and a next item register, connected to supply inputs to the arithmetic logic unit, an output of the arithmetic logic unit being connected to the top item register, the top item register being connected to supply inputs to an internal data bus, the internal data bus being bidirectionally connected to a loop counter, the loop counter being connected to a decrementer, the internal data bus being bidirectionally connected to a stack pointer, return stack pointer, mode register and instruction register, the internal data bus being connected to a memory controller, to a Y register of a return push down stack, an X register and a program counter, the Y register, X register and program counter supplying outputs to an internal address bus, the internal address bus supplying inputs to the memory controller and to an incrementer, the incrementer being connected to the internal data bus, the direct memory access central processing unit supplying inputs to the memory controller, the memory controller having an address/data bus and a plurality of control lines for connection to a random access memory.
この発明の更に別の態様において、マイクロプロセッサシステムにおいて後入れ先出しスタックを動作させる方法が、ラッチとして構成された第１の複数のスタック要素、ランダムアクセスメモリとして構成された第２の複数のリスック要素を提供し、第１および第２の複数のスタック要素をマイクロプロセッサとともに単一の集積回路内に設け、この単一の集積回路の外部のランダムアクセスメモリとして構成された第３の複数のスタック要素を提供し、後入れ先出しスタックに項目を記憶させ、第２の複数のスタック要素にアクセスすることなく第１の複数のスタック要素から第１の複数の項目を取り出し、第１の複数のスタック要素が空であるときは第２の複数のスタック要素から第１の複数の項目を	In a further aspect of the invention, in a microprocessor system, a method for operating a push down stack comprises: supplying a first plurality of stack elements configured as latches, a second plurality of stack elements configured as a random access memory, the first and second plurality of stack elements being provided in a single integrated circuit with the processor, supplying a third plurality of stack elements configured as a random access memory external to the single integrated circuit, storing items in the push down stack, fetching a first plurality of items from the first plurality of stack elements without accessing the second plurality of stack elements, fetching a first plurality of items from the second plurality of stack elements when the first plurality of stack elements are empty, fetching the second plurality of items from the second plurality of stack elements without accessing the third plurality of

取り出し、第３の複数のスタック要素にアクセスすることなく第２の複数のスタック要素から第２の複数の項目を取り出し、第２の複数のスタック要素が空であるときは第３の複数のスタック要素から第２の複数の項目を取り出す。	stack elements, and fetching a second plurality of items from the third plurality of stack elements when the second plurality of stack elements are empty.
この発明の上述の目的と関連の目的、利点および特徴は当該技術に精通するものには、以下のこの発明のより詳細な説明を図面を参照して検討することにより、いっそう容易に理解されるであろう。	The attainment of the foregoing and related objects, advantages and features of the invention should be more readily apparent to those skilled in the art, after review of the following more detailed description of the invention with reference to the drawings.
図面の簡単な説明	BRIEF DESCRIPTION OF THE DRAWINGS
図１はこの発明によるマイクロプロセッサを内蔵する集積回路パッケージの外平面図である。	FIG. 1 is an external, plan view of an integrated circuit package incorporating a microprocessor in accordance with the invention.
図２はこの発明によるマイクロプロセッサのブロック図である。	FIG. 2 is a block diagram of a microprocessor in accordance with the invention.
図３は図１および図２のマイクロプロセッサを内蔵するデータ処理装置の一部のブロック図である。	FIG. 3 is a block diagram of a portion of a data processing system incorporating the microprocessor of FIGS. 1 and 2.
図４は図２に示すマイクロプロセッサの部分のより詳細なブロック図である。	FIG. 4 is a more detailed block diagram of a portion of the microprocessor shown in FIG. 2.
図５は図２に示すマイクロプロセッサの他の部分のより詳細なブロック図である。	FIG. 5 is a more detailed block diagram of another portion of the microprocessor shown in FIG. 2.
図６は図３に部分的に示し、また図１－２および図４－５のマイクロプロセッサを内蔵するデータ処理システムの他の部分のブロック図である。	FIG. 6 is a block diagram of another portion of the data processing system shown in part in FIG. 3 and incorporating the microprocessor of FIGS. 1-2 and 4-5.
図７および図８は図３および図６に部分的に示すデータ処理システムのレイアウト図である。	FIGS. 7 and 8 are layout diagrams for the data processing system shown in part in FIGS. 3 and 6.
図９は単一の集積回路上のデータ処理システムにおけるこの発明によるマイクロプロッサの第２の実施例のレイアウト図である。	FIG. 9 is a layout diagram of a second embodiment of a microprocessor in accordance with the invention in a data processing system on a single integrated circuit.
図 10 は図７および図８のデータ処理システムの一部のより詳細なブロック図である。	FIG. 10 is a more detailed block diagram of a portion of the data processing system of FIGS. 7 and 8.
図 11 は図 12 に示すシステム部分の動作を理解するのに有用なタイミング図である。	FIG. 11 is a timing diagram useful for understanding operation of the system portion shown in FIG. 12.
図 12 は図７および図８のデータ処理システムの別の部分のより	FIG. 12 is another more detailed block diagram of a further

詳細なブロック図である。	portion of the data processing system of FIGS. 7 and 8.
図 13 は図２に示すマイクロプロセッサの一部のより詳細なブロック図である。	FIG. 13 is a more detailed block diagram of a portion of the microprocessor shown in FIG. 2.
図 14 は図３および図７－８に示すシステムの一部のより詳細なブロックおよび概略図である。	FIG. 14 is a more detailed block and schematic diagram of a portion of the system shown in FIGS. 3 and 7-8.
図 15 は図 14 に示すシステム部分の動作を理解するのに有用なグラフである。	FIG. 15 is a graph useful for understanding operation of the system portion shown in FIG. 14.
図 16 は図４に示すシステム部分の一部を示すより詳細なブロック図である。	FIG. 16 is a more detailed block diagram showing part of the system portion shown in FIG. 4.
図 17 は図２に示すマイクロプロセッサの一部のより詳細なブロック図である。	FIG. 17 is a more detailed block diagram of a portion of the microprocessor shown in FIG. 2.
図 18 は図 17 に示すマイクロプロセッサ部分の一部のより詳細なブロック図である。	FIG. 18 is a more detailed block diagram of part of the microprocessor portion shown in FIG. 17.
図 19 は図 18 に示すマイクロプロセッサ部分の一部の動作を理解するのに有用な波形図である。	FIG. 19 is a waveform diagram useful for understanding operation of the part of the microprocessor portion shown in FIG. 18.
図 20 は図４に示すシステム部分の別の部分を示すより詳細なブロック図である。	FIG. 20 is a more detailed block diagram showing another part of the system portion shown in FIG. 4.
図 21 は図４に示すシステム部分の別の部分を示すより詳細なブロック図である。	FIG. 21 is a more detailed block diagram showing another part of the system portion shown in FIG. 4.
図 22 および図 23 は図４に示すシステム部分の別の部分を示すより詳細なブロック図である。	FIGS. 22 and 23 are more detailed block diagrams showing another part of the system portion shown in FIG. 4.
発明の詳細な説明	DETAILED DESCRIPTION OF THE INVENTION
概観	Overview
この発明のマイクロプロセッサは、 高実行速度と 低システムコスト に対して最適化された 32 ビットマイクロプロセッサとして実施するのが望ましい。この実施例ではマイクロプロセッサは 20 ドルで 20MIPS のものと考えることができる。このプロセッサの重要な特徴は次の通りである。	The microprocessor of this invention is desirably implemented as a 32-bit microprocessor optimized for: High execution speed, and Low system cost. In this embodiment, the microprocessor can be thought of as 20 MIPS for 20 dollars. Important distinguishing features of the microprocessor are:
20MIPS の実行に低コスト汎用ダイナミック RAM を使用、	Uses low-cost commodity dynamic RAMs to run 20 MIPS,
１メモリサイクル当たり４命令取り出し、	4 instruction fetch per memory cycle,

オンチップ高速ページモードメモリ管理、	On-chip fast page-mode memory management,
外部キャッシュなしで高速実行、	Runs fast without external cache,
インターフェースチップがほとんど不要、	Requires few interfacing chips,
44 ピン SOJ パッケージ内に 32 ビット CPU	32-bit CPU in 44 pin SOJ package
命令セットはほとんどの動作を８ビット命令で指定できるように構成されている。この考え方の利点は次の通りである。	The instruction set is organized so that most operations can be specified with 8-bit instructions. Two positive products of this philosophy are:
プログラムがより小さくなる、	Programs are smaller,
プログラムをより高速に実行することができる。	Programs can execute much faster.
ほとんどのコンピュータシステムでメモリバスが問題点になっている。バスは命令の取り出しとデータの取り出しと記憶に用いられる。単一のメモリバスサイクル中に四つの命令を取り出す能力はデータ処理に対するバスの利用可能性を大きく向上させる。	The problem point in most computer systems is the memory bus. The bus is used to fetch instructions and fetch and store data. The ability to fetch four instructions in a single memory bus cycle significantly increases the bus availability to handle data.
図について、特に図１について説明すると、約 0.8 インチの実際の大きさの約 100 倍の大きさで示した 44 ピンプラスティック無導線チップキャリア中のパッケージ化された 32 ビットマイクロプロセッサを示す。マイクロプロセッサ 50 が 44 ピンパッケージとして設けられるという点は、通常約 200 の入出力（I/O）ピンを有する代表的なマイクロプロセッサと大きく異なっている。マイクロプロセッサ 50 は２、000 万命令／秒（MIPS）の速度とされている。D0－D31 のラベルが付いたアドレスおよびデータライン 52 が、次に説明するマイクロプロセッサ 50 の動作方法の結果速度を犠牲にすることなくアドレス用とデータ用に共用されている。	Turning now to the drawings, more particularly to FIG. 1, there is shown a packaged 32-bit microprocessor in a 44-pin plastic leadless chip carrier, shown approximately 100 times its actual size of about 0.8 inch on a side. The fact that the microprocessor 50 is provided as a 44-pin package represents a substantial departure from typical microprocessor packages, which usually have about 200 input/output (I/O) pins. The microprocessor 50 is rated at 20 million instructions per second (MIPS). Address and data lines 52, also labeled D0-D31, are shared for addresses and data without sacrificing speed as a result of the manner in which the microprocessor 50 operates, as will be explained below.
ダイナミック RAM	Dynamic RAM
低コスト 44 ピンパッケージに加えて、この高性能マイクロプロセッサ 50 のもう一つの特殊な側面は、列アドレスストローブ（RAS）およびカラムアドレスストローブ（CAS）I/O ピン 54 によって示すように、ダイナミックランダムアクセスメモリ（DRAM）と直接動作する点である。マイクロプロセッサ 50 の他の I/O ピンには、V_{DD} ピン 56、V_{SS} ピン 58、出力イネーブルピン 60、書き込みピン 62、クロックピン 64 およびリセットピン 66 がある	In addition to the low cost 44-pin package, another unusual aspect of the high performance microprocessor 50 is that it operates directly with dynamic random access memories (DRAMs), as shown by row address strobe (RAS) and column address strobe (CAS) I/O pins 54. The other I/O pins for the microprocessor 50 include V_{DD} pins 56, V_{SS} pins 58, output enable pin 60, write pin 62, clock pin 64 and reset pin 66.
あらゆる高速コンピュータは動作するには高速で高価なメモリを必要とする。最も高速なスタティック RAM メモリは低速のダイ	All high speed computers require high speed and expensive memory to operate. The highest speed static RAM memories cost as

ナミック RAM の 10 倍ものコストがかかる。このマイクロプロセッサは低コストのダイナミック RAM を高速ページモードで用いるように最適化されている。ページモードダイナミック RAM はコストをかけずにスタティック RAM の性能を提供する。例えば、低コストの 85nsec のダイナミック RAM は高速ページモードで動作するとき 25nsec でアクセスする。マイクロプロセッサチップ上の集積高速ページモード制御はシステムのインターフェースを簡略化し、高速なシステムを実現する。	much as ten times as much as slow dynamic RAMs. This microprocessor has been optimized to use low-cost dynamic RAM in high-speed page-mode. Page-mode dynamic RAMs offer static RAM performance without the cost. For example, low-cost 85 nsec dynamic RAMs access at 25 nsec when operated in fast page-mode. Integrated fast page-mode control on the microprocessor chip simplifies system interfacing and results in a faster system.
マイクロプロセッサ 50 の詳細を図２に示す。マイクロプロセッサ 50 は、マイクロプロセッサ 50 を構成する単一の集積回路中にメイン中央処理装置（CPU）70 と別の直接メモリアクセス（DMA）CPU72 を含む。メイン CPU70 はそれぞれが入力をライン 82 および 84 によって算術論理演算装置（ALU）80 に提供するように接続されたトップ項目レジスタ 76 と次項目レジスタ 78 を有する第１の 16 深後入れ先出しスタック 74 を有する。ALU80 の出力はライン 86 によってトップ項目レジスタ 76 に接続されている。82 のトップ項目レジスタの出力もまたライン 88 によって内部データバス 90 に接続されている。	Details of the microprocessor 50 are shown in FIG. 2. The microprocessor 50 includes a main central processing unit (CPU) 70 and a separate direct memory access (DMA) CPU 72 in a single integrated circuit making up the microprocessor 50. The main CPU 70 has a first 16 deep push down stack 74, which has a top item register 76 and a next item register 78, respectively connected to provide inputs to an arithmetic logic unit (ALU) 80 by lines 82 and 84. An output of the ALU 80 is connected to the top item register 76 by line 86. The output of the top item register at 82 is also connected by line 88 to an internal data bus 90.
ループカウンタ 92 がライン 96 および 98 によって減分器 94 に接続されている。ループカウンタ 92 はライン 100 によって内部データバス 90 に双方向接続されている。スタックポインタ 102、復帰スタックポインタ 104、モードレジスタ 106 および命令レジスタ 108 もまたライン 110、112、114 および 116 によって内部データバス 90 に接続されている。内部データバス 90 はメモリコントローラ 118 とゲート 120 に接続されている。ゲート 120 は復帰後入れ先出しスタック 134 のＸレジスタ 128、プログラムカウンタ 130 およびＹレジスタ 132 への入力をライン 122、124 および 126 上に提供する。Ｘレジスタ 128、プログラムカウンタ 130 およびＹレジスタ 132 は内部アドレスバス 136 への出力をライン 138、140 および 142 上に提供する。内部アドレスバスはメモリコントローラ 118 と増分器 144 への入力を提供する。増分器 144 はＸレジスタ、プログラムカウンタおよびＹレジスタ 132 への入力をライン 146、122、124 および 126 を介して提供する。DMA CPU72 は	A loop counter 92 is connected to a decrementer 94 by lines 96 and 98. The loop counter 92 is bidirectionally connected to the internal data bus 90 by line 100. Stack pointer 102, return stack pointer 104, mode register 106 and instruction register 108 are also connected to the internal data bus 90 by lines 110, 112, 114 and 116, respectively. The internal data bus 90 is connected to memory controller 118 and to gate 120. The gate 120 provides inputs on lines 122, 124, and 126 to X register 128, program counter 130 and Y register 132 of return push down stack 134. The X register 128, program counter 130 and Y register 132 provide outputs to internal address bus 136 on lines 138, 140 and 142. The internal address bus provides inputs to the memory controller 118 and to an incrementer 144. The incrementer 144 provides inputs to the X register, program counter and Y register 132 via lines 146, 122, 124 and 126. The DMA CPU 72 provides inputs to the memory controller 118 on line 148. The memory controller 118 is connected to a RAM (not shown) by address/data bus 150 and control lines 152.

ライン 148 上にメモリコントローラ 118 への入力を提供する。メモリコントローラ 118 はアドレス／データバス 150 と制御ライン 152 によって RAM（図示せず）に接続されている。	
図２はマイクロプロセッサ 50 が簡単なアーキテクチャを有することを示す。従来の RISC マイクロプロセッサの設計ははるかに複雑である。たとえば、SPARC RISC マイクロプロセッサはマイクロプロセッサ 50 の３倍のゲートを有し、Intel 8960 RISC マイクロプロセッサはマイクロプロセッサ 50 の 20 倍のゲートを有する。このマイクロプロセッサの速度はかなりの部分この簡略性に起因する。このアーキテクチャはこの簡略性を得るために後入れ先出しスタックとレジスタ書き込みを用いている。	FIG. 2 shows that the microprocessor 50 has a simple architecture. Prior art RISC microprocessors are substantially more complex in design. For example, the SPARC RISC microprocessor has three times the gates of the microprocessor 50, and the Intel 8960 RISC microprocessor has 20 times the gates of the microprocessor 50. The speed of this microprocessor is in substantial part due to this simplicity. The architecture incorporates push down stacks and register write to achieve this simplicity.
マイクロプロセッサ 50 は集積回路チップ上に設けられる資源を十分に活用するように調整された I/O を内蔵する。オンチップラッチによって同じ I/O 回路を三つの異なることを処理するのに用いることができる。すなわち、速度を少ししか、あるいは全く犠牲にせずから無アドレス指定、列アドレス指定およびデータを処理することができる。この三重バス多重化によって拡張すべきバッファの数、相互接続ラインの数、I/O ピンおよび内部バッファの数を少なくすることができる。	The microprocessor 50 incorporates an I/O that has been tuned to make heavy use of resources provided on the integrated circuit chip. On chip latches allow use of the same I/O circuits to perform three different processes. To wit, no-addressing, column-addressing and data can be processed with a slight to non-existent sacrifice of speed. This triple bus multiplexing results in fewer buffers to expand, fewer interconnection lines, fewer I/O pins and fewer internal buffers.
オンチップ DRAM 制御を備えることによって、スタティック RAM を用いてえることのできる性能と等しい性能を得ることができる。その結果、メモリをほとんどの RISC システムに用いられるスタティック RAM のシステムコストの 1/4 で設けることができる。	The provision of on-chip DRAM control gives a performance equal to that obtained with the use of static RAMs. As a result, memory is provided at 1/4 the system cost of static RAM used in most RISC systems.
マイクロプロセッサ 50 は１メモリサイクルあたり四つの命令を取り出すことができる。命令は８ビットフォーマットであり、これは 32 ビットマイクロプロセッサである。したがって、システム速度はメモリバス帯域幅の４倍である。この能力によってマイクロプロセッサが次の命令を得る速度におけるフォン・ノイマンの障害を突破することが可能になる。この動作モードは後入れ先出しスタックとレジスタアレーの使用によって可能となる。後入れ先出しスタックは二つの出所と一つの行先に対する明示アドレスの従来技術ではなく、暗示アドレスを使用することを可能とする。	The microprocessor 50 can fetch four instructions per memory cycle. The instructions are in an 8-bit format, and this is a 32-bit microprocessor. System speed is therefore 4 times the memory bus bandwidth. This ability enables the microprocessor to break the Von Neumann obstacle of the speed of getting the next instruction. This mode of operation is possible because of the use of a push down stack and register array. The push down stack allows the use of implied addresses, rather than the prior art technique of explicit addresses for two sources and a destination.

ほとんどの命令はマイクロプロセッサ 50 内で 20 ナノ秒で実行される。したがってマイクロプロセッサはパイプライン遅延を用いず 50 のピーク MIPS で命令を実行することができる。これはマイクロプロセッサ 50 中の少数のゲートとこのマイクロプロセッサのアーキテクチャの高度な並行処理の結果である。	Most instructions execute in 20 nanoseconds in the microprocessor 50. The microprocessor can therefore execute instructions at 50 peak MIPS without pipeline delays. This is a result of the small number of gates in the microprocessor 50 and the high degree of parallelism in the architecture of the microprocessor.
図３は I/O ピン 52 からの DRAM150 のアドレス指定に対してマイクロプロセッサ 50 のライン D8－D14 上でカラムアドレスと列アドレスがどのように多重化されるかを示す。DRAM150 は８個あるうちの一つであるが、ここでは説明をわかりやすくするため一つの DRAM150 のみを示す。図示するように、ライン D11－D18 はそれぞれ DRAM150 の列アドレス入力 A0－A8 に接続されている。さらに、ライン D12－D15 は DRAM150 のデータ入力 DQ1－DQ4 に接続されている。出力イネーブル、書き込みおよびカラムアドレスストローブピン 54 はそれぞれライン 152 によって DRAM150 の出力イネーブル、書き込みおよびカラムアドレスストローブ入力に接続されている。列アドレスストローブピン 54 は列アドレスストローブ複合ロジック 154 を介してライン 152 および 158 によって DRAM150 の列アドレスストローブ入力に接続されている。	FIG. 3 shows how column and row addresses are multiplexed on lines D8-D14 of the microprocessor 50 for addressing DRAM 150 from I/O pins 52. The DRAM 150 is one of eight, but only one DRAM 150 has been shown for clarity. As shown, the lines D11-D18 are respectively connected to row address inputs A0-A8 of the DRAM 150. Additionally, lines D12-D15 are connected to the data inputs DQ1-DQ4 of the DRAM 15. The output enable, write and column address strobe pins 54 are respectively connected to the output enable, write and column address strobe inputs of the DRAM 150 by lines 152. The row address strobe pin 54 is connected through row address strobe decode logic 154 to the row address strobe input of the DRAM 150 by lines 152 and 158.
D0－D7 ピン 52（図１）はマイクロプロセッサ 50 が D11－D18 ピン 52 に多重化された列アドレスおよびカラムアドレスを出力中である時にはアイドル状態である。したがって、D0－D7 ピン 52 は右寄せされた I/O が求められるとき入出力に同時に用いることができる。したがって同時アドレス指定および入出力を行うことができる。	D0-D7 pins 52 (FIG. 1) are idle when the microprocessor 50 is outputting multiplexed row addresses and column addresses on D11-D18 pins 52. The D0-D7 pins 52 can therefore simultaneously be used for I/O when right justified I/O is desired. Simultaneous addressing and I/O can therefore be carried out.
図４はマイクロプロセッサがいかにして単一のクロックサイクル中での複数命令取り出しと前方命令取り出しを通じて、DRAM を用いてスタティック RAM を用いる場合と等しい性能を達成できるかを示す。命令レジスタ 108 は 32 ビット内部データバス 90 上の四つの８ビットバイト命令ワード１－４を受け取る。命令レジスタ 108 の四つの命令バイト１－４の位置はバス 172、174、176 および 178 によってマルチプレクサ 170 に接続される。マイクロプログラムカウンタ 180 はライン 182 によってマルチプレクサ 170 に接続される。マルチプレクサ 170 はバス 186 によって復号器	FIG. 4 shows how the microprocessor is able to achieve performance equal to the use of static RAMs with DRAMs through multiple instruction fetch in a single clock cycle and instruction fetch-ahead. Instruction register 108 receives four 8-bit byte instruction words 1-4 on 32-bit internal data bus 90. The four instruction byte 1-4 locations of the instruction register 108 are connected to multiplexer 170 by busses 172, 174, 176 and 178, respectively. A microprogram counter 180 is connected to the multiplexer 170 by lines 182. The multiplexer 170 is connected to decoder 184 by bus 186. The decoder 184 provides internal signals to the rest of the microprocessor 50 on lines 188.

184 に接続される。復号器 184 はライン 188 上にマイクロプロセッサ 50 の他の部分に対する内部信号を提供する。	
各命令バイト１－４の位置の最上位ビット 190 はライン 194 によって４入力復号器 192 に接続される。復号器 192 の出力はライン 196 によってメモリコントローラ 118 に接続される。プログラムカウンタ 130 は内部アドレスバス 136 によってメモリコントローラ 118 に接続され、命令レジスタ 108 は内部データバス 90 によってメモリコントローラ 118 に接続される。アドレス／データバス 198 と制御バス 200 は DRAM150（図３）に接続される。	Most significant bits 190 of each instruction byte 1-4 location are connected to a 4-input decoder 192 by lines 194. The output of decoder 192 is connected to memory controller 118 by line 196. Program counter 130 is connected to memory controller 118 by internal address bus 136, and the instruction register 108 is connected to the memory controller 118 by the internal data bus 90. Address/data bus 198 and control bus 200 are connected to the DRAM 150 (FIG. 3).
動作中において、残りの命令１－４の最上位ビット 190 がマイクプロセッサ 50 のクロックサイクル中に “1″であるとき、待ち行列にはメモリ参照命令はない。ライン 196 上の復号器 192 の出力はメモリコントローラ 118 による他のアクセスを介在させない前方命令取り出しを要求する。命令レジスタ 108 中の現在の命令が実行されている間に、メモリコントローラ 118 はプログラムカウンタ 130 からの四つの命令の次のセットのアドレスを得、その命令セットを得る。現在の命令セットの実行が完了するまでには次の命令セットを命令レジスタにロードする準備ができている。	In operation, when the most significant bits 190 of remaining instructions 1-4 are “1” in a clock cycle of the microprocessor 50, there are no memory reference instructions in the queue. The output of decoder 192 on line 196 requests an instruction fetch ahead by memory controller 118 without interference with other accesses. While the current instructions in instruction register 108 are executing, the memory controller 118 obtains the address of the next set of four instructions from program counter 130 and obtains that set of instructions. By the time the current set of instructions has completed execution, the next set of instructions is ready for loading into the instruction register.
DMA CPU72 の詳細を図５に示す。内部データバス 90 はメモリコントローラ 118 と DMA 命令レジスタ 210 に接続されている。DMA 命令レジスタ 210 はバス 214 によって DMA プログラムカウンタ 212 に、バス 216 によって転送サイズカウンタ 216 に、バス 222 によって時限転送間隔カウンタ 220 に接続されている。DMA 命令レジスタ 210 はまたライン 226 によって DMA I/O および RAM アドレスレジスタ 224 に接続されている。DMA I/O および RAM アドレスレジスタ 224 はメモリサイクル要求ライン 228 とバス 230 によってメモリコントローラ 118 に接続されている。DMA プログラムカウンタ 212 はバス 232 によって内部アドレスバス 136 に接続されている。転送サイズカウンタ 216 はライン 236 と 238 によって実行済み DMA 命令減分器 234 に接続されている。実行済み DMA 命令減分器 234 はメモリサイクル応答ライン 240 上の制御入力を受け取る。転送サイズカウンタ 216 はその計数を終えると、ライン 242 上に DMA プログラムカウンタ 212 への制御信号を提供する。	Details of the DMA CPU 72 are provided in FIG. 5. Internal data bus 90 is connected to memory controller 118 and to DMA instruction register 210. The DMA instruction register 210 is connected to DMA program counter 212 by bus 214, to transfer size counter 216 by bus 216 and to timed transfer interval counter 220 by bus 222. The DMA instruction register 210 is also connected to DMA I/O and RAM address register 224 by line 226. The DMA I/O and RAM address register 224 is connected to the memory controller 118 by memory cycle request line 228 and bus 230. The DMA program counter 212 is connected to the internal address bus 136 by bus 232. The transfer size counter 216 is connected to an executed DMA instruction decrementer 234 by lines 236 and 238. The executed DMA instruction decrementer 234 receives a control input on memory cycle acknowledge line 240. When transfer size counter 216 has completed its count, it provides a control signal to DMA program counter 212 on line 242. Timed transfer interval counter 220 is connected to decrementer 244 by lines 246 and 248. The decrementer 244 receives a control input from a microprocessor system clock on line 250.

時限転送間隔カウンタ 220 はライン 246 および 248 によって減分器 244 に接続されている。減分器 244 はライン 250 上のマイクロプロセッサシステムクロックからの制御入力を受け取る。	
DMA CPU72 はそれ自体を制御し、命令を取り出し実行する能力を有する。これはメイン CPU70（図２）に対して時間の特定される処理のための双対プロセッサとして動作する。	The DMA CPU 72 controls itself and has the ability to fetch and execute instructions. It operates as a coprocessor to the main CPU 70 (FIG. 2) for time specific processing.
図６はデータライン 52 をあるものは入力ラインとなり他のものは出力ラインとなるように再構成することによってマイクロプロセッサ 50 を電気的にプログラム可能なリードオンリーメモリ（EPROM）260 に接続する態様を示す。データライン 52 D0－D7 は EPROM 260 の対応するデータ端子 262 とのデータのやり取りを可能とする。データライン 52 D9－D18 は EPROM 260 のアドレス端子 264 にアドレスを提供する。データライン 52 D19－D31 はマイクロプロセッサ 50 からの入力をメモリおよび I/O 復号ロジック 266 に提供する。RAS 0/1 制御ライン 268 はメモリおよび I/O 復号ロジックがライン 270 上に DRAM RAS 出力を提供するか、それともライン 272 上に EPROM260 のためのカラムイネーブル出力を提供するかを判定するための制御信号を提供する。マイクロプロセッサ 50 のカラムアドレスストローブ端子 60 はライン 274 上に EPROM260 の対応する端子 276 への出力イネーブル信号を提供する。	FIG. 6 shows how the microprocessor 50 is connected to an electrically programmable read only memory (EPROM) 260 by reconfiguring the data lines 52 so that some of them are input lines and others of them are output lines. Data lines 52 D0-D7 provide data to and from corresponding data terminals 262 of the EPROM 260. Data lines 52 D9-D18 provide addresses to address terminals 264 of the EPROM 260. Data lines 52 D19-D31 provide inputs from the microprocessor 50 to memory and I/O decode logic 266. RAS 0/1 control line 268 provides a control signal for determining whether the memory and I/O decode logic provides a DRAM RAS output on line 270 or it provides a column enable output for the EPROM 260 on line 272. Column address strobe terminal 60 of the microprocessor 50 provides an output enable signal on line 274 to the corresponding terminal 276 of the EPROM 260.
図７および図８はマイクロプロセッサ 50、合計２メガバイトの MSM514258－10 型 DRAM 150、Motorola の 50 メガヘルツ水晶発振器クロック 282、I/O 回路 284、および 27256 型 EPROM 260 を内蔵するワンカードデータ処理システム 280 の前後を示す。I/O 回路 284 は 74HC04 型高速 16 進インバータ回路 286、IDT39C828 型 10 ビット反転バッファ回路 288、IDT39C822 型 10 ビット反転レジスタ回路 290、および二つの IDT39C823 型９ビット非反転レジスタ回路 292 を含む。カード 280 はさらに最大 12V 型直流－直流変換器回路 294、34 ピン２アンプ型ヘッダ 296、同軸雌電源コネクタ 298、および３ピンアンプ直角ヘッダ 300 を加えてなる。カード 280 はより大きなシステムに内蔵するかあるいは内部開発ツールとして用いることのできる低コストの埋め込み可能な製品である。	FIGS. 7 and 8 show the front and back of a one-card data processing system 280 incorporating the microprocessor 50, MSM514258-10 type DRAMs 150 totaling 2 megabytes, a Motorola 50 megahertz crystal oscillator clock 282, I/O circuits 284 and a 27256 type EPROM 260. The I/O circuits 284 include a 74HC04 type high speed hex inverter circuit 286, an IDT39C828 type 10-bit inverting register circuit 288, an IDT39C822 type 10-bit inverting register circuit 290, and two IDT39C823 type 9-bit non-inverting register circuits 292. The card 280 is completed with a maximum 12V type DC-DC converter circuit 294, 34-pin dual AMP type headers 296, a coaxial female power connector 298, and a 3-pin AMP right angle header 300. The card 280 is a low cost, embeddable product that can be incorporated in larger systems or used as an internal development tool.

マイクロプロセッサ 50 はダイナミック RAM と非常に密接に働くように設計された非常に高性能（50MHz）な RISC 型 32 ビット CPU である。マイクロプロセッサ 50 は単一 CPU 構成で可能な理論上の性能限界に近づくものである。最終的にはマイクロプロセッサ 50 もその他のいかなるプロセッサもバス帯域幅とバスパスの数によって制約される。クリティカルパスは CPU とメモリの間にある。	The microprocessor 50 is a very high performance (50 MHz) RISC-type 32-bit CPU designed to work closely with dynamic RAM. The microprocessor 50 approaches the theoretical performance limits possible with a single CPU configuration. Eventually, the microprocessor 50 and any other processor is limited by the bus bandwidth and the number of bus paths. The critical conduit is between the CPU and memory.
バス帯域幅とバスパスの問題に対する解決法の一つに CPU をメモリチップ上に直接集積してすべてのメモリに CPU への直接バスを設ける方法がある。図９は単一の集積回路 312 中に１メガビットの DRAM311 と一体に設けた別のマイクロプロセッサ 310 を示す。この発明以前にはこの解決法は実用的なものではなかった。それはほとんどの高性能 CPU はそれ自体で 500、000 から１、000、000 個のトランジスタと非常に大きなダイサイズを要するためである。マイクロプロセッサ 310 は図１－８のマイクロプロセッサ 50 と等価である。マイクロプロセッサ 50 と 310 はデュアルプロセッサ 70 および 72（図２）あるいは 314 および 316（より小さいメモリ）用に 50、000 個以下のトランジスタしか必要としない現在あるものの中で最もトランジスタ効率の高い高性能 CPU である。マイクロプロセッサ 50 および 310 の非常に高い速度は、ある程度までは能動素子の数が少ないことの結果である。基本的には珪素が少ないほど、電子はそれが目指す場所により速く到着することができる。	One solution to the bus bandwidth/bus path problem is to integrate a CPU directly onto the memory chips, giving every memory a direct bus to the CPU. FIG. 9 shows another microprocessor 310 that is provided integrally with 1 megabit of DRAM 311 in a single integrated circuit 312. Until the present invention, this solution has not been practical. This is because most high performance CPUs require from 500,000 to 1,000,000 transistors and enormous die sizes just by themselves. The microprocessor 310 is equivalent to the microprocessor 50 in FIGS. 1-8. The microprocessors 50 and 310 are the most transistor efficient high performance CPUs in existence, requiring fewer than 50,000 transistors for dual processors 70 and 72 (FIG. 2) or 314 and 316 (smaller memory). The very high speed of the microprocessors 50 and 310 is to a certain extent a result of the small number of active devices. In essence, the less silicon there is, the faster the electrons can get where they are going.
したがって、マイクロプロセッサ 310 はメモリチップダイ 312 上での集積に適した唯一の CPU である。基本マイクロプロセッサ 50 に DRAM アレー311 への近接を利用する簡単な変更を加えることによってもマイクロプロセッサ 50 のクロック速度を 50%あるいはそれ以上増大させることができる。	The microprocessor 310 is therefore the only CPU suitable for integration on the memory chip die 312. Some simple modifications to the basic microprocessor 50 to take advantage of the proximity to the DRAM array 311 can also increase the microprocessor 50 clock speed by 50%, and probably more.
DRAM ダイ 312 上のマイクロプロセッサ 310 のコアは自動系から周辺制御までの幅広いアプリケーションに必要な速度と機能のほとんどを提供する。しかし、集積 CPU 310/DRAM 311 の考え方は多重プロセッサによる解決法によって非常に多量の計算を要する問題を解決する方法を大きく変革する可能性を有するものである。CPU 310/DRAM 311 の組合せはフォン・ノイマンの障害を多数の	The microprocessor 310 core on board the DRAM die 312 provides most of the speed and functionality required for a large group of applications from automatic systems to peripheral control. However, the integrated CPU 310/DRAM 311 concept has the potential to redefine significantly the way multiprocessor solutions can solve a spectrum of very compute intensive problems. The CPU 310/DRAM 311 combination eliminates the Von Neumann obstacle by distributing it across numerous

CPU/DRAM チップ 312 に分散することによって排除する。マイクロプロセッサ 310 は多重処理を行うための特に良好な核である。なぜなら、これは SDI ターゲッティングアレーを念頭に設計されており、プロセッサ間の効率的な通信に対する備えがなされているためである。	CPU/DRAM chips 312. The microprocessor 310 is a particularly good core for multiprocessing. The reason why is because it was designed with the SDI targeting array in mind, and provisions were made for efficient interprocessor communications.
従来の多重プロセッサの実施は CPU の能力を十分に生かすことができないことに加えて、非常に高価であった。多重プロセッサシステムは通常多数の基板レベルあるいはボックスレベルのコンピュータから構成される。その結果、非常に多量のハードウエアとそれに対応する配線、電力消費、および通信上の問題が発生するのが常である。システムが相互接続されるまでに、インターフェースを得るためだけのためにバス速度の 50%もが使用される。	Traditional multiprocessor implementations have been very expensive in addition to being unable to exploit fully the available CPU horsepower. Multiprocessor systems have typically been built up from numerous board level or box level computers. The result is usually an immense amount of hardware with corresponding wiring, power consumption and communications problems. By the time the systems are interconnected, as much as 50% of the bus speed has been utilized just getting through the interfaces.
さらに、多重プロセッサシステムのソフトウエアは数が少ない。多重プロセッサシステムは、一つの CPU に多くの仕事をさせ他はアイドル状態にするシステムソフトウエア中の不適切な負荷分担によって簡単に故障することがある。近年、システムソフトウエアは大きく進歩し、UNIX V.4 をも多重処理を支援するように改善することができる。DUAL Systems や UNISOFT といったメーカーの製品には 68030 型マイクロプロセッサシステム上で信頼性のあるはたらきをするものもある。	Moreover, multiprocessor system software has been scarce. A multiprocessor system can easily be crippled by an inadequate load-sharing algorithm in the system software, which allows one CPU to do a great deal of work and the others to be idle. Great strides have been made recently in systems software, and even UNIX V.4 may be enhanced to support multiprocessing. Several commercial products from such manufacturers as DUAL Systems and UNISOFT do a credible job on 68030 type microprocessor systems now.
マイクロプロセッサ 310 のアーキテクチャはインターフェース上の摩擦のほとんどを解消する。これは最大 64 個の CPU 130/RAM 311 プロセッサはバッファやラッチを用いず相互通信を行うことができるためである。チップ 312 はそれぞれが約 40MIPS の列速度を有する。これは、DRAM 311 を CPU 310 の隣に配置することによってマイクロプロセッサ 310 の命令サイクルをマイクロプロセッサ 50 の半分にすることができるためである。これらのチップ 312 の 64 チップアレーは他の既存のいかなるコンピュータよりも強力である。このようなアレーは３×５カード上に載り、価格はファックシミリ機より低く、小型テレビと同じ電力を消費する。	The microprocessor 310 architecture eliminates most of the interface friction. This is because up to 64 CPU 310/RAM 311 processors should be able to intercommunicate without buffers or latches. Each chip 312 has about 40 MIPS raw speed. This is because placing the DRAM 311 next to the CPU 310 allows the microprocessor 310 instruction cycle to be cut in half, compared to the microprocessor 50. A 64 chip array of these chips 312 is more powerful than any other existing computer. Such an array fits on a 3×5 card, cost less than a FAX machine, and draw about the same power as a small television.
価格対性能比の劇的な変化は常に既存のアプリケーションに変化をもたらし、また新しいアプリケーションを生み出す。共通総称並行処理アルゴリズムは重畳／高速フーリエ変換（FFT）／パタ	Dramatic changes in price/performance always reshape existing applications and almost always create new applications. A common generic parallel processing algorithm handles convolution/Fast Fourier

ーン認識を処理する。集積回路 312 を用いて可能な製品で興味深いものとしては、高速読み取り機、リアルタイム音声認識、音声語翻訳、リアルタイムロボットビジョン、顔から人間を識別する製品、および自動車や航空機の衝突回避システムなどがある。	Transform (FFT)/pattern recognition. Interesting product possibilities using the integrated circuit 312 include high speed reading machines, real-time speech recognition, spoken language translation, real-time robot vision, a product to identify people by their faces, and an automotive or aviation collision avoidance system.
高密度テレビ（HDTV）画像を改善する、あるいは HDTV 情報をより小さな帯域幅に圧縮するためのリアルタイムプロセッサは非常に実現の可能性の高いものである。HDTV における負荷分担は非常に簡単である。色とフレームにしたがってタスクを分割するには６、９あるいは 12 のプロセッサが必要となる。実際にはマイクロプロセッサ 310 に一体化された４メガ RAM が必要となる。	A real time processor for enhancing high density television (HDTV) images, or compressing the HDTV information into a smaller bandwidth, would be very feasible. The load sharing in HDTV could be very straightforward. Splitting up the task according to color and frame would require 6, 9 or 12 processors. Practical implementation might require 4 meg RAMs integrated with the microprocessor 310.
マイクロプロセッサ 310 は以下のような仕様である。	The microprocessor 310 has the following specifications:
制御ライン	Control Lines
４－パワー／グラウンド	4–POWER/GROUND
１－クロック	1–CLOCK
32－データ I/O	32–DATA I/O
４－システム制御	4–SYSTEM CONTROL
外部メモリ取り出し	EXTERNAL MEMORY FETCH
外部メモリ取り出し 自動増分 X	EXTERNAL MEMORY FETCH AUTOINCREMENT X
外部メモリ取り出し 自動減分 Y	EXTERNAL MEMORY FETCH AUTOINCREMENT Y
外部メモリ書き込み	EXTERNAL MEMORY WRITE
外部メモリ書き込み 自動増分 X	EXTERNAL MEMORY WRITE AUTOINCREMENT X
外部メモリ書き込み 自動減分 Y	EXTERNAL MEMORY WRITE AUTOINCREMENT Y
外部 PROM 取り出し	EXTERNAL PROM FETCH
すべてのＸレジスタをロード	LOAD ALL X REGISTERS
すべてのＹレジスタをロード	LOAD ALL Y REGISTERS
すべての PC レジスタをロード	LOAD ALL PC REGISTERS
ＸとＹの入れ替え	EXCHANGE X AND Y
命令取り出し	INSTRUCTION FETCH
PC に追加	ADD TO PC
Ｘに追加	ADD TO X
写像レジスタ書き込み	WRITE MAPPING REGISTER
写像レジスタ読み出し	READ MAPPING REGISTER

レジスタ構成	REGISTER CONFIGURATION
マイクロプロセッサ 310 CPU316 コア	MICROPROCESSOR 310 CPU 316 CORE
カラムラッチ１（1024 ビット）32×32MUX	COLUMN LATCH1 (1024 BITS) 32×32 MUX
スタックポインタ（16 ビット）	STACK POINTER (16 BITS)
カラムラッチ２（1024 ビット）32×32MUX	COLUMN LATCH2 (1024 BITS) 32×32 MUX
スタックポインタ（16 ビット）	STACK POINTER (16 BITS)
プログラムカウンタ 32 ビット	PROGRAM COUNTER 32 BITS
X0 レジスタ 32 ビット（オンチップアクセスに対してのみ起動）	X0 REGISTER 32 BITS (ACTIVATED ONLY FOR ON-CHIP ACCESSES)
Y0 レジスタ 32 ビット（オンチップアクセスに対してのみ起動）	Y0 REGISTER 32 BITS (ACTIVATED ONLY FOR ON-CHIP ACCESSES)
ループカウンタ 32 ビット	LOOP COUNTER 32 BITS
DMA CPU 314 コア	DMA CPU 314 CORE
DMA プログラムカウンタ 24 ビット	DMA PROGRAM COUNTER 24 BITS
命令レジスタ 32 ビット	INSTRUCTION REGISTER 32 BITS
I/O および RAM アドレスレジスタ 32 ビット	I/O AND RAM ADDRESS REGISTER 32 BITS
転送サイズカウンタ 12 ビット	Transfer Size Counter 12 Bits
インターバルカウンタ 12 ビット	Interval Counter 12 Bits
基本チップ 312 のメモリ拡張を提供するために、インテリジェント DRAM を生成することができる。このチップは、プログラムカウンタ、Ｘレジスタおよびｙレジスタの三つのオンチップアドレスレジスタを備えることによって集積回路 312 との高速動作に対して最適化される。その結果、インテリジェント DRAM にアクセスするためにはアドレスを必要とせず、総アクセスサイクルは 10nsec と短くすることができる。それぞれの拡張 DRAM は三つのレジスタのコピーを保持し、そのメモリアドレスを指定するためのコードによって識別される。この三つのレジスタへの増分あるいは追加は実際にはメモリチップ上で発生する。最大 64 のインテリジェント DRAM 周辺装置によって、マルチプレクサやバッファの導入によって速度を犠牲にすることなく大型システムの作成が可能となる。	To offer memory expansion for the basic chip 312, an intelligent DRAM can be produced. This chip will be optimized for high speed operation with the integrated circuit 312 by having three on-chip address registers: the program counter, X register and the Y register. As a result, to access the intelligent DRAM, no address is required, and a total access cycle could be as short as 10 nsec. Each expansion DRAM would maintain its own copy of the three registers and would be identified by a code specifying its memory address. Incrementing and adding to the three registers will actually take place on the memory chips. A maximum of 64 intelligent DRAM peripherals would allow a large system to be created without sacrificing speed by introducing multiplexers or buffers.
マイクロプロセッサ 310 とマイクロプロセッサ 50 の間にはある	There are certain differences between the microprocessor 310 and

種の相異点があり、これはマイクロプロセッサ 310 を DRAM 311 と同じダイ 312 上に設けることに起因する。DRAM 311 を一体化すると既存のオンチップ DRAM 311 回路の活用を可能とするようなマイクロプロセッサ 310 のアーキテクチャ上の変更が可能となる。列とカラムの設計はメモリアーキテクチャに備わっている。DRAM 311 はまず 1024 ビットの列を選択し、それをカラムラッチに記憶し、次にそれらのビットのうちの一つを読み出しあるいは書き込みすべきデータとして選択することによってメモリアレー中のランダムなビットにアクセスする。	the microprocessor 50 that arise from providing the microprocessor 310 on the same die 312 with the DRAM 311. Integrating the DRAM 311 allows architectural changes in the microprocessor 310 logic to take advantage of existing on-chip DRAM 311 circuitry. Row and column design is inherent in memory architecture. The DRAMs 311 access random bits in a memory array by first selecting a row of 1024 bits, storing them into a column latch, and then selecting one of the bits as the data to be read or written.
データのアクセスに要する時間は列アクセスとカラムアクセスに分けられる。カラムラッチに記憶済みのデータの選択はランダムなビットの選択より少なくとも６倍高速である。マイクロプロセッサ 310 は多数のカラムラッチを作成し、それらをキャッシュおよびシフトレジスタとして用いることによってこの高速性を活用する。新しい情報の列を選択することはその結果得られる大きなバス帯域幅によって 1024 ビットの読み出しおよび書き込みを実行することと考えることができる。	The time required to access the data is split between the row access and the column access. Selecting data already stored in a column latch is faster than selecting a random bit by at least a factor of six. The microprocessor 310 takes advantage of this high speed by creating a number of column latches and using them as caches and shift registers. Selecting a new row of information may be thought of as performing a 1024-bit read or write with the resulting immense bus bandwidth.
1. マイクロプロセッサ 50 はその 32 ビットの命令レジスタ 108（図２および図４参照）を四つの８ビット命令のためのキャッシュとして取り扱う。DRAM 311 はカラムビットに対して 1024 ビットラッチを保持するため、マイクロプロセッサ 310 はこのカラムラッチを 128 の８ビット命令のためのキャッシュとして取り扱う。したがって、ほとんど常に次の命令がすでにキャッシュの中に存在している。キャッシュ内の長いループもまた可能であり、マイクロプロセッサ 50 中の四つの命令ループより有用である。	1. The microprocessor 50 treats its 32-bit instruction register 108 (see FIGS. 2 and 4) as a cache for four 8-bit instructions. Since the DRAM 311 maintains a 1024-bit latch for the column bits, the microprocessor 310 treats the column latch as a cache for 128 8-bit instructions. Therefore, the next instruction will almost always be already present in the cache. Long loops within the cache are also possible and more useful than the 4 instruction loops in the microprocessor 50.
2. マイクロプロセッサ 50 はパラメータスタックとリターンスタック用に二つの 16×32 ビットディープレジスタアレー74 および 134（図２）を用いる。マイクロプロセッサ 310 は二つの別の 1024 ビットカラムラッチを作成して、二つの 32×32 ビットアレーに等価なものを提供する。これはレジスタアレーの倍の速度でアクセスすることができる。	2. The microprocessor 50 uses two 16×32-bit deep register arrays 74 and 134 (FIG. 2) for the parameter stack and the return stack. The microprocessor 310 creates two other 1024-bit column latches to provide the equivalent of two 32×32-bit arrays. These can be accessed twice as fast as a register array.
3. マイクロプロセッサ 50 はビデオシフトレジスタへの入出力に用いることのできる DMA 能力を有する。マイクロプロセッサ 310 は	3. The microprocessor 50 has a DMA capability which can be used for I/O to a video shift register. The microprocessor 310 uses yet another

さらに別の 1024 ビットカラムラッチを、CRT 表示装置を直接駆動するための長いビデオシフトレジスタとして用いる。カラー表示に対しては、三つのオンチップシフトレジスタを用いることもできる。これらのシフトレジスタは画素を最大 100MHz で転送することができる。	1024-bit column latch as a long video shift register to drive a CRT display directly. For color displays, three on-chip shift registers could also be used. These shift registers can transfer pixels at a maximum of 100 MHz.
4. マイクロプロセッサ 50 は外部の 32 ビットバスを介してメモリにアクセスする。マイクロプロセッサ 310 のためのメモリ 311 のほとんどは同じダイ 312 上にある。外部からより多くのメモリにアクセスするには８ビットバスを用いて行う。その結果マイクロプロセッサ 50 に比べてダイ、パッケージ、および電力消費が小さくなる。	4. The microprocessor 50 accesses memory via an external 32-bit bus. Most of the memory 311 for the microprocessor 310 is on the same die 312. External access to more memory is made using an 8-bit bus. The result is a smaller die, smaller package and lower power consumption than the microprocessor 50.
5. マイクロプロセッサ 50 はその動作電力の約３分の１を I/0 ピンとその関連のキャパシタンスの充電および放電に用いる。マイクロプロセッサ 50 に接続された DRAM150（図８）は I/0 ドライバ内でその電力のほとんどを消費する。マイクロプロセッサ 310 システムの消費電力はマイクロプロセッサ 50 の約 10 分の１である。これはプロセッサ 310 に隣接して DRAM 311 を有することによって充電および放電すべき外部キャパシタンスのほとんどが除去されるためである。	5. The microprocessor 50 consumes about a third of its operating power charging and discharging the I/O pins and associated capacitances. The DRAMs 150 (FIG. 8) connected to the microprocessor 50 dissipate most of their power in the I/O drivers. A microprocessor 310 system will consume about one-tenth the power of a microprocessor 50. This is because having the DRAM 311 next to the processor 310 eliminates most of the external capacitances to be charged and discharged.
6. 多重処理とはこの解決法の速度を上げるために計算タスクを多数のプロセッサの間で分担することを意味する。多重処理の普及は、現在の個々のプロセッサの費用、さらにプロセッサ間の通信能力上の限度から制約されている。マイクロプロセッサ 310 は多重プロセッサの有望な候補である。これはチップ 312 がメモリ付きの単一のコンピュータであり、低コストと物理的なコンパクトさが実現されているためである。	6. Multiprocessing means splitting a computing task between numerous processors in order to speed up the solution. The popularity of multiprocessing is limited by the expense of current individual processors as well as the limited interprocessor communications ability. The microprocessor 310 is an excellent multiprocessor candidate. This is because the chip 312 is a monolithic computer complete with memory, rendering it low-cost and physically compact.
ビデオ出力を行うためにマイクロプロセッサ 310 に実施されるシフトレジスタもまたプロセッサ間通信リンクとして構成することができる。INMOS トランスピュータは同様な戦略を意図したものであったが、速度ははるかに低く、またマイクロプロセッサ 310 のカラムラッチアーキテクチャに備わっている性能上の利点を有していない。シリアル入出力は多くの多重プロセッサ構成においては多数の隣合うプロセッサが通信を行うため不可欠であ	The shift registers implemented with the microprocessor 310 to perform video output can also be configured as interprocessor communication links. The INMOS transputer attempted a similar strategy, but at much lower speed and without the performance benefits inherent in the microprocessor 310 column latch architecture. Serial I/O is a prerequisite for many multiprocessor topologies because of the many neighbor processors which communicate. A cube has 6 processors. Each processor communicates using these lines:

る。一つのキューブに六つのプロセッサがある。各プロセッサは次のラインを用いて通信する。	
DATA IN CLOCK IN READY FOR DATA DATA OUT DATA READY? CLOCK OUT	DATA IN CLOCK IN READY FOR DATA DATA OUT DATA READY? CLOCK OUT
各プロセッサのオンチップ DRAM 311 を初期化するのに特殊なスタートアップシーケンスが用いられる。	A special start up sequence is used to initialize the on-chip DRAM 311 in each of the processors.
マイクロプロセッサ 310 のカラムラッチアーキテクチャは隣合うプロセッサが内部レジスタあるいは他のチップ 312 の命令キャッシュにさえ直接情報を送ることを可能にする。この技術は既存のプロセッサには用いられていない。これは緊密に結合されたDRAM システムにおいてのみ性能を向上させるためである。	The microprocessor 310 column latch architecture allows neighbor processors to deliver information directly to internal registers or even instruction caches of other chips 312. This technique is not used with existing processors. This is because it only improves performance in a tightly coupled DRAM system.
7. マイクロプロセッサ 50 のアーキテクチャは二種類のループ構造を提供する。すなわち、LOOP－IF－DONE と、MICRO－LOOP である。前者はループアドレスへの入口点の記述に８ビットから 24 ビットのオペランドを要する。後者はループ全体を四つの命令待ち行列内で実行し、ループ入口点はこの待ち行列中の第１の命令を意味する。この待ち行列中のループは外部命令取り出しを行うことなく動作し、最大で長いループ構成体の三倍の速度で実行する。マイクロプロセッサ 310 はこのわずかな違いの二つの構成対を保持する。マイクロプロセッサ 310 のマイクロループは、待ち行列の長さが 1024 ビットあるいは 128 の８ビット命令である点を除いてマイクロプロセッサ 50 の動作と同様に機能する。したがって、マイクロプロセッサ 310 のマイクロループは四つの８ビット命令マイクロプロセッサ 50 の待ち行列では不可能なジャンプ、分岐、呼び出しおよび即時操作を含みうるものである。	7. The microprocessor 50 architecture offers two types of looping structures: LOOP-IF-DONE and MICRO-LOOP. The former takes an 8-bit to 24-bit operand to describe the entry point to the loop address. The latter executes an entire loop within the four instruction queues and the loop entry point is implied as the first instruction in the queue. These loops within the queue run without external instruction fetches and execute up to three times as fast as the long loop structure. The microprocessor 310 retains both structures with a few differences. The microprocessor 310 microloop functions in the same fashion as the microprocessor 50 operation, except the queue is 1024 bits or 128 8-bit instructions long. The microprocessor 310 microloop can therefore contain jumps, branches, calls and immediate operations not possible in the four 8-bit instruction microprocessor 50 queues.
マイクロプロセッサ 50 のマイクロループは簡単なブロック移動と比較機能を実行できるだけである。より大きなマイクロプロセッサ 310 の待ち行列はデジタル信号処理あるいは浮動小数点アルゴリズムの全体がこの待ち行列の中で高速でループすることを可	Microloops in the microprocessor 50 can only perform simple block move and compare functions. The larger microprocessor 310 queue allows entire digital signal processing or floating point algorithms to loop at high speed in the queue.

能にする。	
マイクロプロセッサ 50 は実行の再指令を行う次の四つの命令を提供する。	The microprocessor 50 offers four instructions to redirect execution:
CALL BRANCH BRANCH－IF－ZERO LOOP－IF－NOT－DONE	CALL BRANCH BRANCH-IF-ZERO LOOP-IF-NOT-DONE
これらの命令は８、16、あるいは 24 ビットの長さの可変長アドレスオペランドを取る。マイクロプロセッサ 50 の次アドレスロジックはこの三つのオペランドを現在のプログラムカウンタに加算あるいは減算することによってこれらを同様に取り扱う。マイクロプロセッサ 310 については、16 および 24 ビットオペランドはマイクロプロセッサ 50 の 16 および 24 ビットオペランドと同様に機能する。この８ビットクラスのオペランドはこの命令待ち行列中で全体が動作するように反転される。したがって、次のアドレス決定を速く行うことができる。これはアドレスの 32 ビットではなく 10 ビットだけが関係するためである。10 ビット以降では桁上げも借りも発生しない。	These instructions take a variable length address operand 8, 16 or 24 bits long. The microprocessor 50 next address logic treats the three operands similarly by adding or subtracting them to the current program counter. For the microprocessor 310, the 16 and 24-bit operands function in the same manner as the 16 and 24-bit operands in the microprocessor 50. The 8-bit class operands are reversed to operate entirely within the instruction queue. Next address decisions can therefore be made quickly. This is because only 10 bits of addresses are affected, rather than 32. There is no carry or borrow generated past the 10 bits.
8. マイクロプロセッサ 310 の CPU 316 は狭い DRAM ダイ 312 上に載る。チップサイズを可能な限り小さくするため、マイクロプロセッサ 50 の DMA プロセッサ 72 をより一般的な DMA コントローラ 314 に置き換えた。DMA はマイクロプロセッサ 310 に用いて次の機能を実行する。	8. The microprocessor 310 CPU 316 resides on an already crowded DRAM die 312. To keep chip size as small as possible, the DMA processor 72 of the microprocessor 50 has been replaced with a more typical DMA controller 314. DMA is used with the microprocessor 310 to perform the following functions:
CRT へのビデオ出力	Video output to a CRT
多重プロセッサのシリアル通信	Multiprocessor serial communications
８ビットパラレル入出力	8-bit parallel I/O
DMA コントローラ 314 はシリアル転送とパラレル転送の両方を同時に維持することができる。以下の DMA の出所および行先がマイクロプロセッサ 310 によって支援される。	The DMA controller 314 can maintain both serial and parallel transfers simultaneously. The following DMA sources and destinations are supported by the microprocessor 310:

説明	入出力	ライン
1. ビデオシフトレジスタ	出力	1から3
2. 多重プロセッサ シリアル	両方	チャンネルあたり6ライン
3. 8ビットパラレル	両方	8データ、4制御

DESCRIPTION	I/O	LINES
1. Video shift register	OUTPUT	1 to 3
2. Multiprocessor serial	BOTH	6 lines/channel
3. 8-bit parallel	BOTH	8 data, 4 control

三つの出所は別の 1024 ビットバッファと別の入出力ピンを用いる。したがって、この三つはすべて干渉し合うことなく同時に活動状態になりうる。	The three sources use separate 1024-bit buffers and separate I/O pins. Therefore, all three may be active simultaneously without interference.
マイクロプロセッサ 310 は、単一の多重プロセッサシリアルバッファあるいは各チャンネルのための別個の送受信バッファのいずれかを用いて実施することができ、これによって６個のプロセッサ間での同時双方向通信が可能である。	The microprocessor 310 can be implemented with either a single multiprocessor serial buffer or separate receive and sending buffers for each channel, allowing simultaneous bidirectional communications with six processors simultaneously.
図 10 と図 11 はマイクロプロセッサ 50 に用いられる PROM DMA の詳細を示す。マイクロプロセッサ 50 の実行は最高速の PROM 以外のいかなるものより速い。PROM はマイクロプロセッサ 50 のシステムにおいてプログラムの部分あるいはプログラム全体を記憶するのに用いられる。マイクロプロセッサ 50 は低コストの低速 PROM から高速 DRAM への実行のためのプログラムのローディングを可能とする電源投入時の機能を提供する。この機能を実行するロジックは DMA メモリコントローラ 118 の一部である。この動作は DMA に近いが同一ではない。これは四つの８ビットバイトをマイクロプロセッサ 50 上にアセンブルし、次に DRAM 150 に書き込まなければならないためである。	FIGS. 10 and 11 provide details of the PROM DMA used in the microprocessor 50. The microprocessor 50 executes faster than all but the fastest PROMs. PROMs are used in a microprocessor 50 system to store program segments and perhaps entire programs. The microprocessor 50 provides a feature on power-up to allow programs to be loaded from low-cost, slow speed PROMs into high speed DRAM for execution. The logic which performs this function is part of the DMA memory controller 118. The operation is similar to DMA, but not identical. This is because four 8-bit bytes must be assembled on the microprocessor 50 chip, then written to the DRAM 150.
マイクロプロセッサ 50 は三重多重化データおよびアドレスバス 350 を介して DRAM 150 と直接インターフェースする。この三重多重化データおよびアドレスバス 350 は RAS アドレス、CAS アドレスおよびデータを搬送する。一方、EPROM 260 は非多重化バスによって読み出される。したがって、マイクロプロセッサ 50 はデータおよびアドレスラインの多重化を解き EPROM データの８ビットを読み出す特殊モードを有する。四つの８ビットバイトはこのようにして読み出される。多重化データバス 350 は再びオンされ、	The microprocessor 50 directly interfaces to DRAM 150 over a triple multiplexed data and address bus 350. This triple multiplexed data and address bus 350 carries RAS addresses, CAS addresses and data. The EPROM 260, on the other hand, is read with non-multiplexed busses. The microprocessor 50 therefore has a special mode which unmultiplexes the data and address lines to read 8 bits of EPROM data. Four 8-bit bytes are read in this fashion. The multiplexed bus 350 is turned back on, and the data is written to the DRAM 150.

DRAM 150 にデータが書き込まれる。	
マイクロプロセッサ 50 が RESET 状態を検出すると、プロセッサはメイン CPU 70 を停止させ、モードO（PROM LOAD）命令を DMA CPU 72 命令レジスタに送る。この DMA 命令はメモリコントローラに EPROM 260 のデータをメモリの通常のアクセス時間の８倍で読み出すように指令する。50MHz のマイクロプロセッサ 50 を想定すると、これは 320nsec のアクセス時間を意味する。この命令はまた次のことを示す。	When the microprocessor 50 detects a RESET condition, the processor stops the main CPU 70 and sends a mode O (PROM LOAD) instruction into the DMA CPU 72 instruction register. The DMA instruction directs the memory controller to read the EPROM 260 data at 8 times the normal access time for memory. Assuming a 50 MHz microprocessor 50, this means an access time of 320 nsec. The instruction also indicates:
ロードすべき EPROM 260 の選択アドレス、	The selection address of the EPROM 260 to be loaded,
転送すべき 32 ビットワードの数、	The number of 32-bit words to transfer,
転送先の DRAM 150 のアドレス、	The DRAM 150 address to transfer into.
一つの 32 ビットワードを EPROM 260 から DRAM 150 に転送する動作のシーケンスは次の通りである。	The sequence of activities to transfer one 32-bit word from EPROM 260 to DRAM 150 are:
1. RAS が 352 でローになり、上位アドレスビットからの EPROM 260 選択情報をラッチする。	1. RAS goes low at 352, latching the EPROM 260 select information from the high order address bits.
2. （通常 DRAM CAS アドレスと呼ばれるものからなる）12 のアドレスビットと２バイトの選択ビットが EPROM260 のアドレスピンに行くバス 350 に乗せられる。これらの信号は EPROM 260 からのデータがマイクロプロセッサ 50 に読み込まれるまでライン上に残る。第１のバイトについては、バイト選択ビットは２値的には 00 である。	2. Twelve address bits (consisting of what is normally called DRAM CAS addresses) and 2 byte select bits are placed on the bus 350 going to the EPROM 260 address pins. These signals will remain on the lines until the data from the EPROM 260 has been read into the microprocessor 50. For the first byte, the byte select bits will be binary 00.
3. CAS が 354 でローになり、EPROM 260 のデータを外部アドレス／データバス 350 の下位の８ビット上にイネーブルする。このサイクルのこの部分においては外部アドレス／データバス 350 の下位の８ビットは入力として機能するが、バスの残りの部分は依然として出力として働いていることを認識することが重要である。	3. CAS goes low at 354, enabling the EPROM 260 data onto the lower 8 bits of the external address/data bus 350. It is important to recognize that, during this part of the cycle, the lower 8 bits of the external data/address bus 350 are functioning as inputs, but the rest of the bus is still acting as outputs.
4. マイクロプロセッサ 50 はこれらの八つの最下位ビットを内部的にラッチし、それらを８ビット左にシフトしてそれらを次の有意バイト位置にシフトする。	4. The microprocessor 50 latches these eight least significant bits internally and shifts them 8 bits left to shift them to the next significant byte position.
5. バイトアドレス 01 でステップ２、３および４が繰り返される。ラッチし、それらを８ビット左にシフトし	5. Steps 2, 3 and 4 are repeated with byte address 01. Latch and shift them 8 bits left.
6. バイトアドレス 10 でステップ２、３および４が繰り返される。ラッチし、それらを８ビット左にシフトし	6. Steps 2, 3 and 4 are repeated with byte address 10. Latch and shift them 8 bits left.

7. バイトアドレス 11 でステップ２、３および４が繰り返される。	7. Steps 2, 3 and 4 are repeated with byte address 11.
8. CAS が 356 でハイになり、EPROM 260 をデータバスから取り外す。	8. CAS goes high at 356, taking the EPROM 260 off the data bus.
9. CAS が 358 でハイになり、EPROM 260 をアクセスの終了を示す。	9. CAS goes high at 358, indicating the end of the EPROM 260 access.
10. RAS が 360 でローになり、上位アドレスビットからの DRAM 選択情報をラッチする。同時に、RAS アドレスビットが DRAM 150 にラッチされる。DRAM 150 が選択される。	10. RAS goes low at 360, latching the DRAM select information from the high order address bits. At the same time, the RAS address bits are latched into the DRAM 150. The DRAM 150 is selected.
11. CAS が 362 でローになり、DRAM 150 CAS アドレスをラッチする。	11. CAS goes low at 362, latching the DRAM 150 CAS addresses.
12. マイクロプロセッサ 50 が前にラッチされた EPROM 260 の 32 ビットデータを外部アドレス／データバス 350 に乗せる。Wが 364 でローになり、この 32 ビットを DRAM 150 に書き込む。	12. The microprocessor 50 places the previously latched EPROM 260 32-bit data onto the external address/data bus 350. W goes low at 364, writing the 32 bits into the DRAM 150.
13. Wが 366 でハイになる。CAS が 368 でハイになる。この過程が次のワードに対して続けられる。	13. W goes high at 366. CAS goes high at 368. The process continues with the next word.
図 12 はマイクロプロセッサ 50 のメモリコントローラ 118 の詳細を示す。動作中には、バス要求はそれらがサービスされるまで存在し続ける。CPU 70 の要求には 370 において、1:パラメータスタック、2:リターンスタック、3:データ取り出し、4:命令取り出しの順に優先順位が付けられる。その結果得られる CPU 要求信号および DMA 要求信号はバス要求として 374 でバス許可信号を提供するバス制御 372 に供給される。内部アドレスバス 136 と DMA カウンタ 376 はマルチプレクサ 378 に入力を提供する。列アドレスとカラムアドレスのいずれかがマルチプレクサ 378 からの出力である多重化されたアドレスバス 380 への出力として提供される。多重化アドレスバス 380 と内部データバス 90 はそれぞれマルチプレクサ 382 へのアドレス入力とデータ入力を提供する。シフトレジスタ 384 はライン 390 および 392 上にマルチプレクサ 386 への列アドレスストローブ（RAS）１と二つの制御信号を、またマルチプレクサ 388 へのカラムアドレスストローブ（CAS）１と二つの制御信号を供給する。シフトレジスタ 384 もまた出力イネーブル（OE）信号と書き込み（W）信号をライン 394 および 396 上に、	FIG. 12 shows details of the microprocessor 50 memory controller 118. In operation, bus requests stay present until they are serviced. CPU 70 requests are prioritized at 370 in the order of 1: parameter stack, 2: return stack, 3: Data Fetch, 4: Instruction Fetch. The resulting CPU request signal and a DMA request signal are supplied as bus requests to bus control 372, which provides a bus grant signal at 374. Internal address bus 136 and a DMA counter 376 provide inputs to a multiplexer 378. Either a row address or a column address is provided as an output to multiplexed address bus 380 as an output from the multiplexer 378. The multiplexed address bus 380 and the internal data bus 90 provide address and data inputs, respectively, to multiplexer 382. Shift register 384 supplies row address strobe (RAS) 1 and two control signals to multiplexer 386 and column address strobe (CAS) 1 and two control signals to multiplexer 388 on lines 390 and 392. The shift register 384 also supplies output enable (OE) and write (W) signals on lines 394 and 396 and a control signal on line 398 to multiplexer 382. The shift register 384 receives a RUN signal on line 400 to generate a memory cycle and supplies a MEMORY READY signal on line 402 when an access is complete.

また制御信号をマルチプレクサ 382 へのライン 398 上に供給する。シフトレジスタ 384 はライン 400 上の RUN 信号を受取り、メモリサイクルを発生し、アクセスが完了したときライン 402 上に MEMORY READY 信号を供給する。	
スタック／レジスタアーキテクチャ	Stack/Register Architecture
ほとんどのマイクロプロセッサは変数の一時記憶用にオンチップレジスタを用いる。オンチップレジスタはオフチップ RAM より高速にデータにアクセスする。一時記憶にオンチップ後入れ先出しスタックを用いるマイクロプロセッサはごくわずかである。	Most microprocessors use on-chip registers for temporary storage of variables. The on-chip registers access data faster than off-chip RAM. A few microprocessors use an on-chip push down stack for temporary storage.
スタックは出所レジスタと行先レジス他の選択を不要とするため、オンチップレジスタに比べ高速に動作するという利点を有する。（数理動作あるいは論理動作には常に上の二つのスタック項目が出所として、またスタックの上部が行先として用いられる。）このスタックの欠点はある種の動作が繁雑になることである。特に、ある種のコンパイラアクティビティでは効率上オンチップレジスタを必要とする。	A stack has the advantage of faster operation compared to on-chip registers by avoiding the necessity to select source and destination registers. (A math or logic operation always uses the top two stack items as source and the top of stack as destination.) The stack's disadvantage is that it makes some operations complicated. Some compiler activities in particular require on-chip registers for efficiency.
図 13 に示すように、マイクロプロセッサ 50 はオンチップレジスタ 134 とスタック 74 の両方を提供し、この両者の利点を有する。	As shown in FIG. 13, the microprocessor 50 provides both on-chip registers 134 and a stack 74 and reaps the benefits of both.
利点：	Benefits:
1. スタックの演算とロジックの速度は等価なレジスタだけを有する機械でえられる速度の二倍である。ほとんどのプログラマーおよび最適化コンパイラはこの機能を利用することができる。	1. Stack math and logic is twice as fast as those available on an equivalent register-only machine. Most programmers and optimizing compilers can take advantage of this feature.
2. 16 のレジスタをローカル変数の記憶に用いることができ、ローカル変数は計算を行うためにスタックに転送することができる。変数のアクセス速度は厳密なスタック機械で可能な速度の三倍から四倍である。	2. Sixteen registers are available for on-chip storage of local variables which can transfer to the stack for computation. The accessing of variables is three to four times as fast as available on a strictly stack machine.
スタック 74/レジスタ 134 の組合せアーキテクチャはこれまで最適化コンパイラおよび転送と数理／ロジック命令の混合をコンピュータの設計者たちが的確に理解していなかったために用いられなかった。	The combined stack 74/register 134 architecture has not been used previously due to inadequate understanding by computer designers of optimizing compilers and the mix of transfer versus math/logic instructions.

適応メモリコントローラ	ADAPTIVE MEMORY CONTROLLER
マイクロプロセッサは小さなメモリ構成あるいは大きなメモリ構成で機能するように設計しなければならない。データライン、アドレスラインおよび制御ラインに加えられるメモリ負荷が増えるにつれて、信号のスイッチング速度は低下する。マイクロプロセッサ 50 はアドレス／データバスを三様に多重化し、したがって位相間のタイミングが重要である。この問題の対する従来のアプローチは、バスの位相の間に大きな時間マージンを設けてシステムがそれに接続された少数あるいは多数のメモリチップとともに機能するようにするというものである。この場合速度を 50%も犠牲にしなければならない。	A microprocessor must be designed to work with small memory configurations or large memory configurations. As more memory loads are added to the data lines, address lines, and control lines, the switching speed of the signals slows down. The microprocessor 50 multiplexes the address/data bus three ways, so timing between the phases is critical. A traditional approach to the problem allocates a wide margin of time between bus phases so that systems will work with small or large numbers of memory chips connected. A speed sacrifice of as much as 50% is required.
図 14 に示すように、マイクロプロセッサ 50 はフィードバック技術を用いて、プロセッサがメモリバスのタイミングを小さな負荷に対しては速く、大きな負荷に対しては遅く調整することを可能とする。マイクロプロセッサ 50 からの出力イネーブル（OE）ライン 152 は回路基板上のすべてのメモリ 150 に接続されている。マイクロプロセッサ 50 への出力イネーブルライン 152 上のローディングは接続されたメモリ 150 の数に直接関係している。OE152 が読み出しの後どれだけ急速にハイになるかをモニターすることによって、マイクロプロセッサ 50 はいつデータホールド時間が満足されたかを判定し、バス上に次のアドレスを置く。	As shown in FIG. 14, the microprocessor 50 uses a feedback technique to allow the processor to adjust memory bus timing to be fast with small loads and slower with large loads. The output enable (OE) line 152 from the microprocessor 50 is connected to all memories 150 on the circuit board. The loading on the output enable line 152 to the microprocessor 50 is directly related to the number of memories 150 connected. By monitoring how rapidly 0E 152 goes high after a read, the microprocessor 50 is able to determine when the data hold time has been satisfied and place the next address on the bus.
OE ライン 152 のレベルはマイクロプロセッサのメモリコントローラへのライン上に内部 READY 信号を発生する CMOS 入力バッファ 410 によってモニターされる。図 15 のグラフのカーブ 414 および 416 は負荷の軽いメモリシステムと負荷の大きいメモリシステムに見られる立ち上がり時間の相違を示す。OE ライン 152 が READY 信号を発生する所定のレベルに達したとき、ドライバ 418 は OE ライン 152 上に出力イネーブル信号を発生する。	The level of the OE line 152 is monitored by CMOS input buffer 410 which generates an internal READY signal on line 412 to the microprocessor’s memory controller. Curves 414 and 416 of the FIG. 15 graph show the difference in rise time likely to be encountered from a lightly loaded memory system to a heavily loaded memory system. When the OE line 152 has reached a predetermined level to generate the READY signal, driver 418 generates an output enable signal on OE line 152.
命令キャッシュ中のスキップ	Skip Within The Instruction Cache
マイクロプロセッサ 50 は各メモリサイクルで四つの８ビット命令を取り出し、図 16 に示すようにそれらを 32 ビット命令レジスタ 108 に記憶する。あるクラスの“テストおよびスキップ”命令	The microprocessor 50 fetches four 8-bit instructions each memory cycle and stores them in a 32-bit instruction register 108, as shown in FIG. 16. A class of “test and skip” instructions can very rapidly

は四つの命令キャッシュ内で非常に高速なジャンプ動作を非常に急速に実行することができる。	execute a very fast jump operation within the four instruction cache.
スキップ条件：	Skip Conditions:
always	always
ACC はゼロではない	ACC non-zero
ACC は負	ACC negative
桁上げフラグ＝論理１	Carry flag = logic 1
never	never
ACC＝ゼロ	ACC = zero
ACC は正	ACC positive
桁上げフラグ＝論理０	Carry flag = logic zero
スキップ命令の位置は 32 ビット命令レジスタ 108 中の四つのバイト位置のどれであってもよい。テストが成功である場合、スキップは命令レジスタ 108 中の残りの一つ、二つあるいは三つの８ビット命令を飛び越え、次の四つの命令のグループのレジスタ 108 へのローディングを発生させる。図示するように、このスキップ動作はライン 422 上で２ビットマイクロ命令カウンタ 180 をゼロにリセットし、同時に次の命令グループをレジスタ 108 にラッチすることによって実施することができる。命令レジスタ中のスキップに続くいかなる命令も新しい命令によって重ね書きされ実行されない。	The skip instruction can be located in any of the four byte positions in the 32-bit instruction register 108. If the test is successful, skip will jump over the remaining one, two, or three 8-bit instructions in the instruction register 108 and cause the next four-instruction group to be loaded into the register 108. As shown, the skip operation is implemented by resetting the 2-bit microinstruction counter 180 to zero on line 422 and simultaneously latching the next instruction group into the register 108. Any instructions following the skip in the instruction register are overwritten by the new instructions and not executed.
スキップの利点は最適化コンパイラおよびスマートプログラマがより長い条件つきのジャンプ命令の代わりにそれを頻繁に用いることができるという点である。スキップはまたループがカウントダウンするとき、あるいはスキップが次の命令グループにジャンプするとき出るマイクロループを可能とする。その結果非常に高速なコードが得られる。	The advantage of skip is that optimizing compilers and smart programmers can often use it in place of the longer conditional jump instruction. Skip also makes possible microloops which exit when the loop counts down or when the skip jumps to the next instruction group. The result is very fast code.
（PDP－８や Data General の NOVA といった）他の機械は単一の命令をスキップする能力を提供する。マイクロプロセッサ 50 は最大で三つの命令をスキップする能力を提供する。	Other machines (such as the PDP-8 and Data General NOVA) provide the ability to skip a single instruction. The microprocessor 50 provides the ability to skip up to three instructions.
命令キャッシュ中のマイクロループ	Microloop In The Instruction Cache
マイクロプロセッサ 50 は命令レジスタ 108 中の一つから三つの	The microprocessor 50 provides the microloop instruction to

命令を繰り返し実行するためのマイクロループ命令を提供する。このマイクロループ命令は内部データバス 90 に接続されたループカウンタ 92（図２）と連動して機能する。マイクロループを実行するために、プログラムがループカウンタ 92 中の計数値を記憶する。マイクロループは命令レジスタ 108 の第１、第２、第３あるいは最終バイト 420 に入れることができる。第１の位置に入れた場合、実行によってループカウンタ 92 に記憶された数にマシンサイクルを掛けたものに等しい遅延が作成されるだけである。第２、第３あるいは最終バイト 420 に入れられた場合、マイクロループ命令が実行されるとループ計数値がゼロであるかどうかがテストされる。ゼロである場合、次の命令が継続して実行される。ゼロでない場合、ループカウンタ 92 が減分され、２ビットマイクロ命令カウンタがクリアされ、命令レジスタ中の前の命令が再度実行される。	execute repetitively from one to three instructions residing in the instruction register 108. The microloop instruction works in conjunction with the loop counter 92 (FIG. 2) connected to the internal data bus 90. To execute a microloop, the program stores a count in loop counter 92. Microloop may be placed in the first, second, third, or last byte 420 of the instruction register 108. If placed in the first position, execution will just create a delay equal to the number stored in loop counter 92 times the machine cycle. If placed in the second, third, or last byte 420, when the microloop instruction is executed, it will test the loop count for zero. If zero, execution will continue with the next instruction. If not zero, the loop counter 92 is decremented and the 2-bit microinstruction counter is cleared, causing the preceding instructions in the instruction register to be executed again.
マイクロループはブロック移動と探索動作に有用である。命令レジスタ 108 の外への完全なブロック移動を実行することによって、移動速度が二倍になる。これはメモリサイクルが命令取り出しと共用されるのではなく、そのすべてが移動に用いられるためである。このようなマイクロループのハードウエアによる実施態様は、相当する機能を有する従来のソフトウエアによる実施よりはるかに高速である。	Microloop is useful for block move and search operations. By executing a block move completely out of the instruction register 108, the speed of the move is doubled. This is because all memory cycles are used by the move rather than being shared with instruction fetching. Such a hardware implementation of microloops is much faster than conventional software implementation of a comparable function.
最適 CPU クロックスキーム	Optimal CPU Clock Scheme
高速マイクロプロセッサの設計者は広い温度範囲、広い電圧揺動範囲、および半導体加工の大きなばらつきに対しても動作する製品を作らなければならない。温度、電圧および加工はすべてトランジスタの伝搬遅延に影響する。従来、CPU の設計はこの三つのパラメータが悪い場合に回路が定格クロック速度で動作するように成される。その結果、設計は論理上の最高性能の半分の速度でクロックしなければならないようなものになり、悪い条件でも適切に動作する。	The designer of a high speed microprocessor must produce a product which operates over wide temperature ranges, wide ranges of voltage fluctuation, and wide variations in semiconductor processing. Temperature, voltage, and process all affect transistor propagation delays. Traditional CPU designs are done so that with the worse case of the three parameters, the circuit will operate at the rated clock speed. The results are designs that must be clocked at a speed half of that of their maximum theoretical performance, so they will operate properly in worse case conditions.
マイクロプロセッサ 50 は図 17－図 19 に示す技術を用いてシステムクロックとそれに要する位相を発生する。クロック回路 430	The microprocessor 50 uses the technique shown in FIGS. 17-19 to generate the system clock and its required phases. Clock circuit 430 is the known “ring oscillator” used to test process performance. The clock is

は処理性能をテストするのに用いられる周知の“リングオシレータ”である。クロックはマイクロプロセッサ 50 の他の部分と同じシリコンチップ上に製作される。	fabricated on the same silicon chip as the rest of the microprocessor 50.
リングオシレータの周波数は温度、電圧および加工のパラメータによって決まる。室温では周波数は 100MHz 近辺である。摂氏 70 度では速度は 50MHz である。リングオシレータ 430 はシステムクロックとして有用であり、その段 431 は図 19 に示す位相０－位相３の出力 433 を生成する。これはその性能が同じシリコンダイ上の他のすべてのトランジスタに同じ影響を及ぼすパラメータを反映するためである。リングオシレータ 430 からシステムタイミングを発生することによって、CPU 70 は常に可能な最大周波数で動作するが速くなりすぎることはない。たとえば、あるダイの加工が不良でトランジスタ速度が低い場合、マイクロプロセッサ 50 のラッチとゲートは通常より低速で動作する。マイクロプロセッサ 50 のリングオシレータクロック 430 はラッチやゲートと同じダイ上の同じトランジスタから作られるため、これもまた低速で動作し（低い周波数で発振する）、チップのロジックの残りが適正に動作することを可能とする補償を提供する。	The ring oscillator frequency is determined by the parameters of temperature, voltage, and process. At room temperature, the frequency will be in the neighborhood of 100 MHz. At 70 degrees Centigrade, the speed will be 50 MHz. The ring oscillator 430 is useful as a system clock, with its stages 431 producing phase O-phase 3 outputs 433 shown in FIG. 19. This is because its performance tracks the parameters which similarly affect all other transistors on the same silicon die. By deriving system timing from the ring oscillator 430, CPU 70 will always execute at the maximum frequency possible, but never too fast. For example, if the processing of a particular die is not good resulting in slow transistors, the latches and gates on the microprocessor 50 will operate slower than normal. Since the microprocessor 50 ring oscillator clock 430 is made from the same transistors on the same die as the latches and gates, it too will operate slowly (oscillating at a lower frequency), providing compensation which allows the rest of the chip’s logic to operate properly.
非同期／同期 CPU	Asynchronous/Synchronous CPU
ほとんどのマイクロプロセッサはすべてのシステムタイミングを一つのクロックから発生する。この場合の問題点は、システムの異なる部分がすべての動作の速度を低下させる可能性があることである。マイクロプロセッサ 50 は図 17 に示すようなデュアルクロックスキームを提供し、CPU 70 はメモリコントローラ 118（図２）の一部をなす I/O インターフェース 432 と非同期に動作し、I/O インターフェース 432 はメモリと I/O 装置の外界と同期して動作する。CPU 70 は適応リングカウンタクロック 430 を用いて可能な最高の速度で動作する。速度は温度、電圧および加工に応じて四倍に変化する。ビデオ表示更新やディスクドライブ読み出しおよび書き込み等の動作については外界はマイクロプロセッサ 50 に同期していなければならない。この同期は I/O インターフェース 432 によって行われ、I/O インターフェース 432 の速度は	Most microprocessors derive all system timing from a single clock. The disadvantage is that different parts of the system can slow all operations. The microprocessor 50 provides a dual-clock scheme as shown in FIG. 17, with the CPU 70 operating asynchronously to I/O interface 432 forming part of memory controller 118 (FIG. 2) and the I/O interface 432 operating synchronously with the external world of memory and I/O devices. The CPU 70 executes at the fastest speed possible using the adaptive ring counter clock 430. Speed may vary by a factor of four depending upon temperature, voltage, and process. The external world must be synchronized to the microprocessor 50 for operations such as video display updating and disc drive reading and writing. This synchronization is performed by the I/O interface 432, and the speed of the I/O interface 432 is controlled by a conventional crystal clock 434. The interface 432 processes requests for memory accesses from the microprocessor 50 and acknowledges the presence of I/O data. The

従来の水晶クロック 434 によって制御される。インターフェース 432 はマイクロプロセッサ 50 からのメモリアクセス要求を処理し、I/O データの存在を確認する。マイクロプロセッサ 50 は一つのメモリサイクル中で最大で四つの命令を取り出し、別のメモリアクセスを要求する前に非常に有用な作業を行うことができる。CPU 70 の可変速度を I/O インターフェース 432 の固定速度から減結合することによって、それぞれが最適な性能を達成することができる。CPU 70 とインターフェース 432 の再結合はライン 436 上のハンドシェーク信号によって行われ、データ／アドレスはバス 90、136 を通過する。	microprocessor 50 fetches up to four instructions in a single memory cycle and can perform much useful work before requiring another memory access. By decoupling the variable speed of the CPU 70 from the fixed speed of the I/O interface 432, optimum performance can be achieved by each. Recoupling between the CPU 70 and the interface 432 is accomplished with handshake signals on lines 436, with data/addresses passing on bus 90, 136.
DRAM チップに埋め込まれた非同期／同期 CPU	Asynchronous/Synchronous CPU Imbedded On A DRAM Chip
DRAM 311 と CPU 314（図９）が同じダイに配置されるとき、システム性能はさらに増強される。トランジスタの近接性は DRAM 311 と CPU 314 のパラメータが互いに密接に追従しあうことを意味する。室温では、CPU 314 が 100MHz で動作するだけではなく、DRAM 311 はそれに対応するに十分な速度でアクセスする。I/O インターフェース 432 によって行われる同期は DMA と I/O ポートの読み出しおよび書き込みのためのものである。システムによっては（たとえば計算機のように）I/O 同期を全く必要としないものもあり、I/O クロックはリングカウンタクロックに結合される。	System performance is enhanced even more when the DRAM 311 and CPU 314 (FIG. 9) are located on the same die. The proximity of the transistors means that DRAM 311 and CPU 314 parameters will closely follow each other. At room temperature, not only would the CPU 314 execute at 100 MHz, but the DRAM 311 would access fast enough. The synchronization performed by the I/O interface 432 would be for DMA and reading and writing I/O ports. In some systems (such as calculators) no I/O synchronization at all would be required, and the I/O clock would be tied to the ring counter clock.
可変幅オペランド	Variable Width Operands
多くのマイクロプロセッサは可変幅オペランドを提供する。マイクロプロセッサ 50 は同じ演算コードを用いて８、16、あるいは 24 ビットのオペランドを処理する。図 20 には 32 ビット命令レジスタ 108 と８ビット命令を選択する２ビットマイクロ命令レジスタ 180 を示す。二つのクラスのマイクロプロセッサ 50 の命令は８ビットより大きい場合がある。すなわち、ジャンプクラスと即時クラスである。ジャンプあるいは即時演算コードは８ビットであるが、オペランドの長さは８、16 あるいは 24 ビットである場合がある。これが起こりうるのは、オペランドは命令レジスタ中で右寄せされなければならないためである。これはオペランドの最下位ビットが常に命令レジスタの最下位ビットに位置することを	Many microprocessors provide variable width operands. The microprocessor 50 processes operands of 8, 16, or 24 bits using the same op-code. FIG. 20 shows the 32-bit instruction register 108 and the 2-bit microinstruction register 180 which selects the 8-bit instruction. Two classes of microprocessor 50 instructions can be greater than 8-bits. To wit, these are the jump class and the immediate class. A jump or immediate op-code is 8 bits, but the operand can be 8, 16, or 24 bits long. This is possible because operands must be right justified in the instruction register. This means that the least significant bit of the operand is always located in the least significant bit of the instruction register. The microinstruction counter 180 selects which 8-bit instruction to execute. If a jump or immediate instruction is decoded, the state of the 2-bit

意味する。マイクロ命令カウンタ 180 は実行すべき８ビット命令を選択する。ジャンプあるいは即時命令が復号されると、２ビットマイクロ命令カウンタの状態が必要な８、16、あるいは 24 ビットのオペランドをデータバスのアドレス上に選択する。選択されなかった８ビットバイトには復号器 440 とゲート 442 の動作によってゼロがロードされる。この技術の利点は他のマイクロプロセッサ中の異なるオペランドサイズを指定するのに必要な演算コードの数を節約できることである。	microinstruction counter selects the required 8, 16, or 24 bit operand onto an address of the data bus. The unselected 8-bit bytes are loaded with zeros by operation of decoder 440 and gates 442. The advantage of this technique is the saving of a number of op-codes required to specify the different operand sizes in other microprocessors.
三倍長スタックキャッシュ	Triple-Length Stack Cache
コンピュータの性能はシステムメモリの帯域幅に直接関係している。メモリが高速であるほどコンピュータも高速である。高速メモリは高価であり、したがって少量の高速メモリをそれが必要とされるメモリアドレスに移動させる技術が開発されている。多量の低速メモリが高速メモリによって恒常的に更新され、それによって大きな高速メモリアレーの外観を呈している。この技術の一般的な実施態様は高速メモリキャッシュとして知られている。このキャッシュはメモリアクセスの衝撃を和らげる高速で動作するショックアブソーバと考えることができる。ショックを吸収できる以上のメモリが必要とされるとき、これは底をつき低速メモリがアクセスされる。大半のメモリ動作はこのショックアブソーバ自体で処理することができる。	Computer performance is directly related to the system memory bandwidth. The faster the memories, the faster the computer. Fast memories are expensive, so techniques have been developed to move a small amount of high-speed memory around to the memory addresses where it is needed. A large amount of slow memory is constantly updated by the fast memory, giving the appearance of a large fast memory array. A common implementation of the technique is known as a high-speed memory cache. The cache may be thought of as fast acting shock absorber smoothing out the bumps in memory access. When more memory is required than the shock can absorb, it bottoms out and slow speed memory is accessed. Most memory operations can be handled by the shock absorber itself.
マイクロプロセッサ 50 のアーキテクチャは上部の二つのスタック位置 76 および 78 に直接結合された ALU 80（図２）を有する。したがってスタック 74 のアクセス時間はプロセッサの実行速度に直接影響を及ぼす。マイクロプロセッサ 50 のスタックアーキテクチャは図 21 に示す三倍長キャッシュ技術に特に適しており、この三倍長キャッシュ技術はオンチップラッチ 450 の速度で動作する大きなスタックメモリの外観を呈する。ラッチ 450 はこのチップ上に設けられる最高速の形態のメモリ素子であり、わずか 3nsec でデータを送る。しかし、ラッチ 450 は多数のトランジスタを必要とする。オンチップ RAM 452 はラッチより少ないトランジスタしか要しないが、速度は５分の１である（15nsec アクセス）。オ	The microprocessor 50 architecture has the ALU 80 (FIG. 2) directly coupled to the top two stack locations 76 and 78. The access time of the stack 74 therefore directly affects the execution speed of the processor. The microprocessor 50 stack architecture is particularly suitable to a triple-length cache technique, shown in FIG. 21 which offers the appearance of a large stack memory operating at the speed of on-chip latches 450. Latches 450 are the fastest form of memory device built on the chip, delivering data in as little as 3 nsec. However latches 450 require large numbers of transistors to construct. On-chip RAM 452 requires fewer transistors than latches, but is one-fifth as fast (15 nsec access). Off-chip RAM 150 is the slowest storage of all. The microprocessor 50 organizes the stack memory hierarchy as three interconnected stacks 450, 452 and 454. The latch stack 450 is the fastest

フチップ RAM 150 は最も低速の記憶装置である。マイクロプロセッサ 50 はスタックメモリの階層を三つの相互接続されたスタック 450、452 および 454 として組織する。ラッチスタック 450 は最高速であり、最も頻繁に用いられる。オンチップ RAM スタック 452 がその次である。オフチップ RAM スタック 454 が最低速である。スタック変調はスタックの有効アクセス時間を決定する。一群のスタック動作が決してスタック上の四つ以上の連続する項目をプッシュもプルもしない場合、動作は 3nsec のラッチスタック内ですべて行われる。四つのラッチ 456 がすべていっぱいになったとき、ラッチスタック 450 の下部にあるデータはオンチップ RAM スタック 452 の上部に書き込まれる。オンチップ RAM スタック 452 中の 16 の位置 458 がいっぱいのとき、オンチップ RAM スタック 452 の下部にあるデータはオフチップ RAM スタック 454 の上部に書き込まれる。いっぱいのスタック 450 からデータを取り出すとき、ラッチスタックポインタ 462 からのスタック空きライン 460 がオンチップ RAM スタック 452 からのデータを転送する前に四回の取り出しが行われる。より低速のオンチップ RAM アクセスを行う前にラッチスタック 450 が空になるのを待つことによって、ラッチ 456 の高い有効速度がプロセッサに利用可能となる。オンチップ RAM スタック 452 とオフチップ RAM スタック 454 に同じ方法が用いられる。	and most frequently used. The on-chip RAM stack 452 is next. The off-chip RAM stack 454 is slowest. The stack modulation determines the effective access time of the stack. If a group of stack operations never push or pull more than four consecutive items on the stack, operations will be entirely performed in the 3 nsec latch stack. When the four latches 456 are filled, the data in the bottom of the latch stack 450 is written to the top of the on-chip RAM stack 452. When the sixteen locations 458 in the on-chip RAM stack 452 are filled, the data in the bottom of the on-chip RAM stack 452 is written to the top of the off-chip RAM stack 454. When fetching data off a full stack 450, four fetches will be performed before stack empty line 460 from the latch stack pointer 462 transfers data from the on-chip RAM stack 452. By waiting for the latch stack 450 to empty before performing the slow on-chip RAM access, the high effective speed of the latches 456 are made available to the processor. The same approach is employed with the on-chip RAM stack 452 and the off-chip RAM stack 454.
多項式発生命令	Polynomial Generation Instruction
多項式はエラー修正、暗号化、データ圧縮、およびフラクタルの発生に有用である。多項式は一連のシフトおよび排他的論理和動作によって発生する。従来技術ではこの目的のために特殊なチップが設けられる。	Polynomials are useful for error correction, encryption, data compression, and fractal generation. A polynomial is generated by a sequence of shift and exclusive OR operations. Special chips are provided for this purpose in the prior art.
マイクロプロセッサ 50 は ALU 80 の動作の態様を少し変更することによって外部ハードウエアを用いることなく高速で多項式を発生することができる。図 21 に示すように、多項式は“順序”（フィードバック項としても知られる）をＣレジスタ 470 にロードすることによって発生する。値 31（32 の繰り返しが起こる）がダウンカウンタ 427 にロードされる。レジスタ 474 にゼロがロー	The microprocessor 50 is able to generate polynomials at high speed without external hardware by slightly modifying how the ALU 80 works. As shown in FIG. 21, a polynomial is generated by loading the “order” (also known as the feedback terms) into C register 470. The value 31 (causing 32 iterations) is loaded into down counter 472. Register 474 is loaded with zero. Register 476 is loaded with the starting polynomial value. When the polynomial instruction executes, C register 470 is

ドされる。レジスタ 476 には開始多項式値がロードされる。多項式命令が実行されると、Ｃレジスタ 470 がＢレジスタ 476 の最下位ビットが１である場合、Ａレジスタ 474 で排他的論理和を取られる。あるいは、Ａレジスタ 474 の内容が変更されずに ALU 80 を通過する。このＡとＢの組合せは次にシフタ 478 と 480 で右シフトされる（２で割られる）。この動作が指定された繰り返し数だけ自動的に繰り返され、その結果得られる多項式がＡレジスタ 474 に残る。	exclusively ORed with A register 474 if the least significant bit of B register 476 is a 1. Or, the contents of the A register 474 passes through the ALU 80 unaltered. The combination of A and B is then shifted right (divided by 2) with shifters 478 and 480. The operation automatically repeats the specified number of iterations, and the resulting polynomial is left in A register 474.
高速乗算	Fast Multiply
ほとんどのマイクロプロセッサは 16×16 あるいは 32×32 の乗算命令を提供する。乗算は順次１ビットあたり１シフト／加算を要する、あるいは 32 ビットのデータに対して 32 サイクルを要する。マイクロプロセッサ 50 は少ないサイクル数を用いて小さな数による乗算を可能とする高速乗算を提供する。図 23 はこの高速アルゴリズムの実施に用いるロジックを示す。乗算を行うには、乗数のサイズー１がダウンカウンタ 472 に入れられる。４ビットの乗数に対しては数３がダウンカウンタ 472 に記憶される。Ａレジスタ 474 にゼロがロードされる。この乗数はビット反転してＢレジスタ 476 に書き込まれる。たとえば、ビット反転された５（２値では 0101）がＢに 1010 として書き込まれる。被乗数がＣレジスタ 470 に書き込まれる。高速乗算命令を実行すると、計数が終了したときＡレジスタ 474 にその結果が残る。この高速乗算命令は多くのアプリケーションにおいて一つの数をそれよりはるかに小さい数でスケーリングするという点で重要である。32×32 ビットの乗算と 32×４ビットの乗算の間の速度の差は８倍である。乗数の最下位ビットが“1″である場合、Ａレジスタ 474 とＣレジスタ 470 の内容が加算される。乗数の最下位ビットが“0″である場合、Ａレジスタ 474 の内容が変更されずに ALU 80 を通過する。ALU 80 の出力は繰り返しのたびにシフタ 482 によって左シフトされる。Ｂレジスタ 476 の内容は繰り返しのたびにシフト 480 によって右シフトされる。	Most microprocessors offer a 16×16 or 32×32 bit multiply instruction. Multiply when performed sequentially takes one shift/add per bit, or 32 cycles for 32 bits of data. The microprocessor 50 provides a high speed multiply which allows multiplication by small numbers using only a small number of cycles. FIG. 23 shows the logic used to implement the high speed algorithm. To perform a multiply, the size of the multiplier -1 is placed in the down counter 472. For a four bit multiplier, the number 3 would be stored in the down counter 472. Zero is loaded into the A register 474. The multiplier is written bit reversed into the B register 476. For example, a bit-reversed 5 (binary 0101) would be written into B as 1010. The multiplicand is written into the C register 470. Executing the fast multiply instruction will leave the result in the A register 474, when the count has been completed. The fast multiply instruction is important because many applications scale one number by a much smaller number. The difference in speed between multiplying a 32×32 bit and a 32×4 bit is a factor of 8. If the least significant bit of the multiplier is a “1” the contents of the A register 474 and the C register 470 are added. If the least significant bit of the multiplier is a “0” the contents of the A register are passed through the ALU 80 unaltered. The output of the ALU 80 is shifted left by shifter 482 in each iteration. The contents of the B register 476 are shifted right by the shifter 480 in each iteration.
命令実行の考え方	INSTRUCTION EXECUTION PHILOSOPHY

マイクロプロセッサ 50 は速度が重要である領域のほとんどにおいて高速Dラッチを用いている。低速のオンチップ RAM は二次記憶装置として用いられている。	The microprocessor 50 uses high speed D latches in most of the speed critical areas. Slow on-chip RAM is used as secondary storage.
マイクロプロセッサ 50 の命令実行の考え方は次のような速度の階層を作成することにある。	The microprocessor 50 philosophy of instruction 30 execution is to create a hierarchy of speed as follows:
ロジックおよびDラッチ転送　１サイクル　20nsec 計算　２サイクル　40nsec 取り出し／記憶オンチップ RAM　２サイクル　40nsec 現在の RAS ページにおける 取り出し／記憶　４サイクル　80nsec RAS サイクルでの 取り出し／記憶　11 サイクル　220nsec	Logic and D latch transfers　1 cycle　20 nsec Math　2 cycles　40 nsec Fetch/store on-chip RAM　2 cycles　40 nsec Fetch store in current RAS page　4 cycles　80 nsec Fetch/store with RAS cycle　11 cycles　220 nsec
50MHz クロックでは、多くの動作を 20nsec で行うことができ、他のほとんどすべての動作を 40nsec で行うことができる。	With a 50 MHz clock, many operations can be performed in 20 nsec and almost every other operation in 40 nsec.
速度を最大化するために、プロセッサ設計におけるある種の技術が用いられた。それには次のものが含まれる。	To maximize speed, certain techniques in processor design have been used. They include:
アドレスに関する算術演算の排除、	Eliminating arithmetic operations on addresses,
１メモリサイクルあたり最大四つの命令取り出し、	Fetching up to four instructions per memory cycle,
パイプラインを用いない命令復号、	Pipelineless instruction decoding
必要となる前に結果を発生すること、	Generating results before they are needed,
３レベルのスタックキャッシングの使用	Use of three level stack caching.
パイプラインの考え方	PIPELINE PHILOSOPHY
コンピュータ命令は通常たとえば取り出し、復号、レジスタ読み出し、実行、および記憶というような順次の断片に細分化される。各断片には一つのマシンサイクルを要する。大半の低減命令セットコンピュータ（RISC）チップにおいて、命令は３サイクルから６サイクルを要する。	Computer instructions are usually broken down into sequential pieces, for example: fetch, decode, register read, execute, and store. Each piece will require a single machine cycle. In most Reduced Instruction Set Computer (RISC) chips, instruction require from 3 cycles to 6 cycles.
RISC 命令は非常に並列的である。たとえば、SPARC（Sun Computer の RISC チップ）の 70 の異なる命令はそれぞれが５サイクルを有する。“パイプライニング”と呼ばれる技術を用いて、連続する命令の異なる段階を重ねることができる。	RISC instructions are very parallel. For example, each of 70 different instructions in the SPARC (SUN Computer’s RISC chip) has 5 cycles. Using a technique called “pipelining”, the different phases of consecutive instructions can be overlapped.
パイプライニングを理解するために５軒の家屋を建てる場合を考えてみる。それぞれの家屋には基礎工事、かまち組、建て入れおよび配線、屋根ぶき、および内装工事を順次行わなければならない。それぞれの工事に一週間かかるものとする。１軒の家屋を	To understand pipelining, think of building five residential homes. Each home will require in sequence, a foundation, framing, leveling and wiring, roofing, and interior finish. Assume that each activity takes one week. To build one house will take five weeks.

建てるには５週間必要である。	
しかし、各部分全体を作りたい場合はどうであろうか。それぞれの作業に従事する作業員グループは一つずつであるが、基礎工事を行うグループが１軒目を終えたときすぐに彼らに２軒目に着工させることができる、他も同様である。５週間後には１軒目が完成しており、さらに５軒分の基礎工事がすでに終わっている。かまち組、建て入れおよび配線、屋根ぶき、内装工事を行う各作業員をすべて作業させていた場合、５週間目以降は毎週１軒ずつ新しい家が完成することになる。	But what if you want to build each part entirely? You have only one of each work crew, but when the foundation men finish on the first house, you immediately start them on the second house, and so on. At the end of five weeks, the first home is complete, but you also have five foundations. If you have kept the framing, leveling and wiring, roofing, and interior guys all busy, from five weeks on, a new house will be completed each week.
SPARC 等の RSIC チップはこのような方法で一つの命令を一つのマシンサイクル中に実行する。実際には、RISC チップは各マシンサイクルで五つの命令のうちの５分の１を実行する。そしてこの五つの命令が順に行われる場合、それぞれのマシンサイクルで一つの命令が完了する。	This is the way a RISC chip like SPARC executes an instruction in a single machine cycle. In reality, a RISC chip is executing one fifth of five instructions each machine cycle. And if five instructions stay in sequence, an instruction will be completed each machine cycle.
パイプラインの問題点はパイプを命令でいっぱいにしておけるかどうかということである。分岐や呼び出しといったシーケンス外の命令が発生するたびに、パイプを次のシーケンスで再度満たさなければならない。この結果生じるパイプラインを再充填するための無駄時間は、IF/THEN/ELSE の文あるいはサブルーチンが多数発生する場合にはかなり大きくなる。	The problems with a pipeline are keeping the pipe full with instructions. Each time an out of sequence instruction such as a branch or call occurs, the pipe must be refilled with the next sequence. The resulting dead time to refill the pipeline can become substantial when many IF/THEN/ELSE statements or subroutines are encountered.
パイプライン法	The Pipeline Approach
マイクロプロセッサ 50 はこのようなパイプラインは持っていない。速度を上げるためにこのマイクロプロセッサので用いている方法は、命令取り出しを前に取り出された命令の実行に重ねて行うというものである。これによって命令（最も一般的なもの）の半分以上が 20nsec の単一のマシンサイクル中に完全に実行される。これが可能なのは次の理由による。	The microprocessor 50 has no pipeline as such. The approach of this microprocessor to speed is to overlap instruction fetching with execution of the previously fetched instructions. Beyond that, over half the instructions (the most common ones) execute entirely in a single machine cycle of 20 nsec. This is possible for the following reasons:
1. 命令復号は 2.5nsec で終了する。	1. Instruction decoding is completed in 2.5 nsec.
2. 増分／減分された値および他の数値はそれらが必要になる前に計算され、その実行にはラッチング信号を要するのみである。	2. Incremented/decremented values and other values are calculated before they are needed, requiring only a latching signal to execute.
3. 低速のメモリは 4nsec でアクセスする高速Ｄラッチによる高速動作から隠されている。	3. Slow memory is hidden from high speed operations by high-speed D latches which access in 4 nsec.

チップの設計がより複雑であるとき、このマイクロプロセッサには問題がある。このチップのユーザーにとっての利点はパイプラインの停止が起こりえないことから最終的なスループットがより高速になることである。使用可能性フラグビットを用いたパイプラインの同期やその他の同様なパイプライン処理はこのマイクロプロセッサには不要である。	This is a problem for this microprocessor when the chip design is more complex. The advantage for the chip user is faster ultimate throughput since pipeline stalls cannot exist. Pipeline synchronization with availability flag bits and other such pipeline handling is not required by this microprocessor.
たとえば、RISC マシンの中には状態ビットをテストする命令が前の命令によてセットされたフラグがテスト可能になるまで最大で４サイクル待たなければならないものもある。ハードウエアおよびソフトウエアデバッギングもいくぶん簡単である。それはユーザーがパイプ中で五つの命令を同時に見る必要がないためである。	For example, in some RISC machines an instruction which tests a status flag may have to wait for up to 4 cycles for the flag set by the previous instruction to be available to be tested. Hardware and software debugging is also somewhat easier. This is because the user doesn't have to visualize five instructions simultaneously in the pipe.
重複する命令取り出し／実行	OVERLAPPING INSTRUCTION FETCH/EXECUTE
マイクロプロセッサ 50 が実行する処理手順の中で再低速のものはメモリへのアクセスである。メモリはデータが読み出されるか書き込まれるときにアクセスされる。メモリはまた命令が取り出されるとき読み出される。マイクロプロセッサ 50 は次の命令の取り出しを前に取り出された命令の実行の背後に隠すことができる。マイクロプロセッサ 50 は命令を４バイトの命令グループとして取り出す。一つの命令グループは一つから四つの命令を含むことができる。命令グループの実行に要する時間量は簡単な命令の場合の４サイクルから乗算の場合の 64 サイクルまでの範囲に及ぶ。	The slowest procedure the microprocessor 50 performs is to access memory. Memory is accessed when data is read or written. Memory is also read when instructions are fetched. The microprocessor 50 is able to hide fetch of the next instruction behind the execution of the previously fetched instructions. The microprocessor 50 fetches instructions in 4-byte instruction groups. An instruction group may contain from one to four instructions. The amount of time required to execute the instruction group ranges from 4 cycles for simple instructions to 64 cycles for a multiply.
新しい命令グループが取り出されると、マイクロプロセッサの命令復号器がその四つのバイトすべての最上位ビットを見る。命令の最上位ビットがメモリアクセスが必要であるかどうかを決定する。例えば、呼び出し、取り出し、および記憶はすべてメモリアクセスの実行を必要とする。四つのバイトすべてが非ゼロの最上位ビットを有する場合、マイクロプロセッサは次の４バイト命令グループのメモリ取り出しを開始する。グループ中の最後の命令の実行が終了すると、次の４バイトの命令グループが実行可能であり、データバス上で待機しており、命令レジスタにラッチさ	When a new instruction group is fetched, the microprocessor instruction decoder looks at the most significant bit of all four of the bytes. The most significant bit of an instruction determines if a memory access is required. For example, call, fetch, and store all require a memory access to execute. If all four bytes have nonzero most significant bits, the microprocessor initiates the memory fetch of the next sequential 4-byte instruction group. When the last instruction in the group finishes executing, the next 4-byte instruction group is ready to be executed and waiting on the data bus, awaiting only to be latched into the instruction register. If the 4-byte instruction group required four or more cycles to execute and the next sequential access was a column address strobe

れるのを待つばかりとなっている。この４バイトの命令グループの実行に四つ以上のサイクルを要し、次の順次のアクセスがカラムアドレスストローブ（CAS）サイクルであった場合、この命令取り出しは完全に実行と重複している。	(CAS) cycle, the instruction fetch was completely overlapped with execution.
内部アーキテクチャ	INTERNAL ARCHITECTURE
マイクロプロセッサ 50 のアーキテクチャは以下のものからなる。	The microprocessor 50 architecture consists of the following:
パラメータスタック↔ ALU* ↔ ←　32ビット　→ 16ディープ 数理および論理演算に使用。 後入れ先出しスタック。 オフチップRAM へのオーバーフローが可能。	Y レジスタ リターンスタック ←　32ビット　→ 16ディープ サブルーチンと割込みリターンアドレスおよびローカル変数に使用。 後入れ先出しスタック。 オフチップRAM へのオーバーフローが可能。 スタックのトップに関しアクセス可能。
PARAMETER STACK <--> ALU* <--> <---32 BITS---> 16 DEEP Used for math and logic operations. Push down stack. Can overflow into off-chip RAM.	Y REGISTER RETURN STACK <---32 BITS---> 16 DEEP Used for subroutine and interrupt return addresses as well as local variables. Push down stack. Can overflow into off-chip RAM. Can also be accessed regarding the top of stack.
ループカウンタ（32 ビット、２で増分可能） テストおよびループ命令のクラスで使用。 Xレジスタ（32 ビット、４で増分あるいは減分可能） RAM 位置の指示に使用。 プログラムカウンタ（32 ビット、４で増分） RAM 中の４バイト命令グループ。 命令レジスタ（32 ビット） ４バイトの命令グループが復号あるいは実行されている間それをホールドする。	LOOP COUNTER (32-bit, can increment by 2) Used by class of test and loop instructions. X REGISTER (32-bit, can increment or decrement by 4). Used to point to RAM locations. PROGRAM COUNTER (32-bit, increments by 4). 4-byte instruction groups in RAM. INSTRUCTION REGISTER (32-bit) Holds 4-byte instruction groups while they are being decoded and executed.

＊数理および論理演算にはオペランドとしてトップ項目およびトップの次のパラメータスタック項目を用いる。その結果はパラメータスタック上にプッシュされる。	*Math and logic operations use the top item and next to top parameter stack items as the operands. The result is pushed onto the parameter stack.
＊サブルーチンからのリターンアドレスがリターンスタックに載せられる。Ｙレジスタは RAM 位置へのポインタとして用いられる。Ｙレジスタはリターンスタックのトップ項目であるため、インデックスのネスティングは簡単である。	*Return addresses from subroutines are placed on the return stack. The Y register is used as a pointer to RAM locations. Since the Y register is the top item of the return stack, nesting of indices is straightforward.
モードーモードおよび状態ビット付きレジスタ	MODE–A register with mode and status bits.
モードービット：	Mode-Bits:
－“1″の場合メモリアクセスを８だけ減速する。 “0″の場合全速で実行。（低速 EPROM へのアクセス用に設けられる。）	- Slow down memory accesses by 8 if “1”. Run full speed if “0”. (Provided for access to slow EPROM.)
－“1″の場合システムクロックを 1023 で割って電力消費を低減する。 “0″の場合全速で実行。（このビットがセットされている場合オンチップカウンタは減速する。）	- Divide the system clock by 1023 if “1” to reduce power consumption. Run full speed if “0”. (On-chip counters slow down if this bit is set.)
－外部割込み１をイネーブルする。 －外部割込み２をイネーブルする。 －外部割込み３をイネーブルする。 －外部割込み４をイネーブルする。 －外部割込み５をイネーブルする。 －外部割込み６をイネーブルする。 －外部割込み７をイネーブルする。	- Enable external interrupt 1. - Enable external interrupt 2. - Enable external interrupt 3. - Enable external interrupt 4. - Enable external interrupt 5. - Enable external interrupt 6. - Enable external interrupt 7.
オンチップメモリ位置：	On-Chip Memory Locations:
モードービット	MODE-BITS
DMAーポインタ	DMA-POINTER
DMAーカウンタ	DMA-COUNTER
スタックーポインタ ーパラメータスタックへのポインタ。	STACK-POINTER–Pointer into parameter stack.
スタックー深さーオンチップパラメータスタックの深さ。	STACK-DEPTH–Depth of on-chip parameter stack
Ｒスタックーポインターリターンスタックへのポインタ。	RSTACK-POINTER–Pointer into return stack
Ｒスタックー深さーオンチップリターンスタックの深さ。	RSTACK-DEPTH–Depth of on-chip return stack
アドレス指定モード高ポイント	Addressing Mode High Points
データバスの幅は 32 ビットである。メモリ取り出しおよびメモ	The data bus is 32 bits wide. All memory fetches and stores are

リ記憶はすべて 32 ビットである。メモリバスアドレスは 30 ビットである。最下位の２ビットはある種のアドレス指定モードにおいて４バイトのうちの１バイトの選択に用いられる。プログラムカウンタ、Ｘレジスタ、およびＹレジスタはＤラッチとして実施され、その出力はメモリアドレスバスとバス増分器／減分器に行く。これらのレジスタのうちの一つの増分は迅速に発生可能である。これは増分された値がすでに増分／減分ロジックを介してリップルされており、ラッチにクラックするだけでよい。分岐および呼び出しは 32 ビットワード境界で行われる。	32-bit. Memory bus addresses are 30-bit. The least significant 2 bits are used to select one-out-of-four bytes in some addressing modes. The program counter, X register, and Y register are implemented as D latches with their outputs going to the memory address bus and the bus incrementer/decrementer. Incrementing one of these registers can happen quickly. This is because the incremented value has already rippled through the increment/decrement logic and needs only be cracked [sic] into the latch. Branches and Calls are made to 32-bit word boundaries.
命令セット	INSTRUCTION SET
32 ビット命令フォーマット	32-Bit Instruction Format
32 ビット命令は CALL、BRANCH、BRANCH－IF－ZERO、LOOP－IF－NOT－DONE である。これらの命令は有効アドレスの計算を必要とする。多くのコンピュータにおいて、有効アドレスは現在のプログラムカウンタに対してオペランドを加算あるいは減算することによって計算される。この数理演算を行うには４から７のマシンサイクルを要し、マシンの実行を確実に停滞させてしまう。このマイクロプロセッサの方法は必要な数理演算をアセンブル時あるいはリンキング時に実行しまたランタイムにおいてはるかに簡単な“次のページへの増分”あるいは“前のページへの減分”動作を行う。その結果、このマイクロプロセッサは一つのサイクル中に実行を分岐する。	The 32-bit instructions are CALL, BRANCH, BRANCH-IF-ZERO, and LOOP-IF-NOT-DONE. These instructions require the calculation of an effective address. In many computers, the effective address is calculated by adding or subtracting an operand with the current program counter. This math operation requires from 4 to 7 machine cycles to perform and can definitely bog down machine execution. The microprocessor’s strategy is to perform the required math operation at assembly or linking time and do a much simpler “increment to next page” or “decrement to previous page” operation at run time. As a result, the microprocessor branches execute in a single cycle.
24 ビットオペランド形式：	24-Bit Operand Form:
バイト 1　　バイト 2　　バイト 3　　バイト 4 WWWWWW XX － YYYYYYYY　－ YYYYYYYY　－ YYYYYYYY	Byte 1　　Byte 2　　Byte 3　　Byte 4 WWWWWW XX -　YYYYYYYY -　YYYYYYYY -　YYYYYYYY
24 ビットオペランドで、現在のページはプログラムカウンタの上位６ビットによって定義される。	With a 24-bit operand, the current page is considered to be defined by the most significant 6 bits of the program counter.
16 ビットオペランド形式：	16-Bit Operand Form:
QQQQQQQQ －WWWWWW XX －YYYYYYYY －YYYYYYYY	QQQQQQQQ–WWWWWW XX–YYYYYYYY—YYYYYYYY
16 ビットオペランドで、現在のページはプログラムカウンタの上位 14 ビットによって定義される。	With a 16-bit operand, the current page is considered to be defined by the most significant 14 bits of the program counter.
８ビットオペランド形式：	8-Bit Operand Form:

QQQQQQQQ －QQQQQQQQ －WWWWWW XX －YYYYYYYY	QQQQQQQQ–QQQQQQQQ–WWWWWW XX—YYYYYYYY
８ビットオペランドで、現在のページはプログラムカウンタの上位 22 ビットによって定義される。	With an 8-bit operand, the current page is considered to be defined by the most significant 22 bits of the program counter.
QQQQQQQQ－任意の８ビット命令	QQQQQQQQ–Any 8-bit instruction.
WWWWWW －命令演算コード	WWWWWW–Instruction op-code.
XX－アドレスビットの使用方法を選択：	XX–Select how the address bits will be used:
00 －すべての上位ビットをゼロにする（ページゼロアドレス指定）	00 - Make all high-order bits zero. (Page zero addressing)
01 －上位ビットをを増分する（次のページを使用）	01 - Increment the high-order bits. (Use next page)
10 －上位ビットを減分する（前のページを使用）	10 - Decrement the high-order bits. (Use previous page)
11 －上位ビットを変更しない（現在のページを使用）	11 - Leave the high-order bits unchanged. (Use current page)
YYYYYYYY－アドレスオペランドフィールド。このフィールドは常に２ビット左シフトされ（バイトアドレスではなくワードを発生し）、プログラムカウンタにロードされる。マイクロプロセッサ命令復号器はこの四つのバイト内の命令演算コードの位置によってこのオペランドフィールドの幅を計算する。	YYYYYYYY - The address operand field. This field is always shifted left 2 bits (to generate a word rather than byte address) and loaded into the program counter. The microprocessor instruction decoder figures out the width of the operand field by the location of the instruction op-code in the four bytes.
コンパイラあるいはアセンブラは通常所望のアドレスに到達するのに必要な最短のオペランドを用い、先行バイトが他の命令をホールドするのに用いることができるようにしている。有効アドレスは次のものを組み合わせることによって計算される。	The compiler or assembler will normally use the shortest operand required to reach the desired address so that the leading bytes can be used to hold other instructions. The effective address is calculated by combining:
現在のプログラムカウンタ、	The current program counter,
命令中の８、16、あるいは 24 ビットアドレスオペランド、四つの可能なアドレス指定モードのうちの一つを用いる。	The 8, 16, or 24 bit address operand in the instruction, Using one of the four allowed addressing modes.
有効アドレス計算の例例	EXAMPLES OF EFFECTIVE ADDRESS CALCULATION
例 1:	Example 1:
バイト 1 バイト 2 バイト 3 バイト 4 QQQQQQQQ QQQQQQQQ 00000011 10011000	Byte 1 Byte 2 Byte 3 Byte 4 QQQQQQQQ QQQQQQQQ 00000011 10011000
バイト１および２の“QQQQQQQQ″は呼び出し命令の前に二つの他の命令の実行をホールドすることのできる４バイトメモリ取り出し中のスペースを示す。バイト３は（11 ビットで示される）現在のページ内の呼び出し命令（000000）を示す。バイト４は 16 進数 98 がプログラムカウンタのビット２からビット 10 に入れられる	The “QQQQQQQQs” in Byte 1 and 2 indicate space in the 4-byte memory fetch which could be hold two other instructions to be executed prior to the call instruction. Byte 3 indicates a call instruction (000000) in the current page (indicated by the 11 bits). Byte 4 indicates that the hexadecimal number 98 will be placed into the program counter bits 2 through 10. (Remember, a call or branch always goes to a word boundary

ことを示す。（呼び出しあるいは分岐は常にワード境界に向かい、したがって二つの最下位ビットは常にゼロにセットされる。）この命令の効果は現在のページのワード位置 16 進数 98 においてサブルーチンを呼び出すことである。プログラムカウンタの上位 22 ビットは現在のページを定義し、変更されない。	so the two least significant bits are always set to zero). The effect of this instruction would be to call a subroutine at word location hexadecimal 98 in the current page. The most significant 22 bits of the program counter define the current page and will be unchanged.
例 2:	Example 2:
バイト 1　　バイト 2　　バイト 3　　バイト 4 000001 01　　00000001　　00000000　　00000000	Byte 1　　Byte 2　　Byte 3　　Byte 4 000001 01　　00000001　　00000000　　00000000
プログラムカウンタが 16 進数 00000156 であったと仮定すると、これは２種で表すと次の通りである。	If we assume that the program counter was hexadecimal 0000 0156 which is binary:
00000000 00000000 00000001 01010110 ＝古いプログラムカウンタ	00000000 00000000 00000001 01010110=Old program counter.
バイト１は分岐命令の演算コード（000001）を示し、また“01″は次のページの選択を表す。バイト２、３、および４はアドレスオペランドである。これらの 24 ビットは左に二つシフトされ、ワードアドレスを定義する。16 進数 0156 を左に二つシフトすると 16 進数 0558 になる。これは 24 ビットオペランド命令であるため、プログラムカウンタの上位６ビットは現在のページを定義する。これらの６ビットは増分されて次のページを選択する。この命令を実行すると、プログラムカウンタに 16 進数 0400 0558 がロードされる。これは２値で表すと次の通りである。	Byte 1 indicates a branch instruction op-code (000001) and “01” indicates select the next page. Bytes 2, 3, and 4 are the address operand. These 24 bits will be shifted to the left two places to define a word address. Hexadecimal 0156 shifted left two places is hexadecimal 0558. Since this is a 24-bit operand instruction, the most significant 6 bits of the program counter define the current page. These 6 bits will be incremented to select the next page. Executing this instruction will cause the program counter to be loaded with hexadecimal 0400 0558. This is represented in binary as:
00000100 00000000 00000101 01011000＝新しいプログラムカウンタ	00000100 00000000 00000101 01011000=New program counter.
命令	Instructions
CALL－LONG	CALL-LONG
0000 00XX －YYYYYYYY－YYYYYYYY－YYYYYYYY	0000 00XX–YYYYYYYY–YYYYYYYY–YYYYYYYY
プログラムカウンタに指定された有効ワードアドレスをロードする。現在の PC 内容をリターンスタック上にプッシュする。	Load the program counter with the effective word address specified. Push the current PC contents onto the return stack.
他の効果：桁あげあるいはモード、効果なし。オンチップリターンスタックがいっぱいである場合、リターンスタックが外部メモリサイクルを強制する。	Other Effects: Carry or modes, no effect. May cause return stack to force an external memory cycle if on-chip return stack is full.
BRANCH	BRANCH

0000 01XX －YYYYYYYY－YYYYYYYY－YYYYYYYY	0000 01XX–YYYYYYYY–YYYYYYYY–YYYYYYYY
プログラムカウンタに指定された有効ワードアドレスをロードする。	Load the program counter with the effective word address specified.
他の効果：なし	Other Effects: None
BRANCH－IF－ZERO	BRANCH-IF-ZERO
0000 10XX －YYYYYYYY－YYYYYYYY－YYYYYYYY	0000 10XX–YYYYYYYY–YYYYYYYY–YYYYYYYY
パラメータスタック上のトップ値をテストする。この値がゼロに等しいとき、プログラムカウンタに指定された有効ワードアドレスをロードする。トップ値がゼロに等しくないとき、プログラムカウンタを増分し、次の命令を取り出し実行する。	Test the top value on the parameter stack. If the value is equal to zero, load the program counter with the effective word address specified. If the top value is not equal to zero, increment the program counter and fetch and execute the next instruction.
他の効果：なし	Other Effects: None
LOOP－IF－NOT－DONE	LOOP-IF-NOT-DONE
0000 11YY－(XXXX XXXX)－(XXXX XXXX)－(XXXX XXXX)	0000 11YY– (XXXX XXXX) – (XXXX XXXX) – (XXXX XXXX)
ループカウンタがゼロでない場合、プログラムカウンタに指定された有効ワードアドレスをロードする。ループカウンタがゼロである場合、ループカウンタを減分し、プログラムカウンタを増分し、次の命令を取り出し実行する。	If the loop counter is not zero, load the program counter with the effective word address specified. If the loop counter is zero, decrement the loop counter, increment the program counter and fetch and execute the next instruction.
他の効果：なし	Other Effects: None
８ビット命令の考え方	8-Bit Instructions Philosophy
マイクロプロセッサ 50 中の作業のほとんどは８ビット命令によって行われる。このマイクロプロセッサで８ビット命令が可能であるのは、暗示スタックアドレス指定を多く用いているためである。32 ビットアーキテクチャの多くは実行すべき動作の指定に８ビットを用いるが、二つの出所と一つの行先を指定するには別の 24 ビットを用いる。	Most of the work in the microprocessor 50 is done by the 8-bit instructions. Eight bit instructions are possible with the microprocessor because of the extensive use of implied stack addressing. Many 32-bit architectures use 8 bits to specify the operation to perform but use an additional 24 bits to specify two sources and a destination.
数理演算および論理演算に関しては、マイクロプロセッサ 50 は出所オペランドをトップスタック項目および次のスタック項目として指定することによって、スタックに固有の利点を生かしている。数理演算および論理演算が行われ、オペランドがスタックから取り出され、結果がスタックに戻される。この結果命令ビットおよびレジスタを非常に有効に活用することができる。このよう	For math operations and logic operations, the microprocessor 50 exploits the inherent advantage of a stack by designating the source operands as the top stack item and the next stack item. The math operation or logic operation is performed, the operands are fetched from the stack, and the result is returned to the stack. The result is a very efficient utilization of instruction bits as well as registers. A comparable situation exists between Hewlett Packard calculators (which use a stack)

な状況が Hewlett Packard の計算機（これはスタックを用いる）とスタックを用いない Texas Instrument の計算機の間に存在する。HP 製品での同じ動作をすると TI の場合のキーストロークの 1/2 か 1/3 のキーストロークしか要しない。	and Texas Instrument calculators which don't. The identical operation on an HP will require 1/2 to 1/3 the keystrokes of the TI.
８ビット命令を利用できることで、さらに別のアーキテクチャ上の技術革新が可能となる。すなわち、単一の 32 ビットメモリサイクルでの四つの命令の取り出しである。複数命令取り出しの利点は次の通りである。	The availability of 8-bit instructions also allows another architectural innovation. To wit, this is the fetching of four instructions in a single 32-bit memory cycle. The advantages of fetching multiple instructions are:
低速メモリを用いても高い実行速度が得られる。	Increased execution speed is obtained even with slow memories.
費用をかけずに Harvard（別のデータバスおよび命令バス）と同様の性能が得られる。	Performance similar to the Harvard (separate data and instruction busses) is obtained without the expense.
命令のグループを最適化する機会がある。	There are opportunities to optimize groups of instructions.
このミニキャッシュ内でループを行うことができる。四つの命令グループ内部のマイクロループは探索とブロック移動に有効である。	It is possible to perform loops within this mini-cache. The microloops inside the four instruction group are effective for searches and block moves.
スキップ命令	Skip Instructions
マイクロプロセッサ 50 は４バイト命令グループと呼ばれる 32 ビットのかたまりの中の命令を取り出す。これらの四つのバイトは四つの８ビット命令あるいは８ビット、16 ビットあるいは 24 ビットの命令が混合したものを含む場合がある。マイクロプロセッサ中のスキップ命令は４倍と命令グループ中の残りの命令すべてをスキップして次の４バイト命令グループを得るメモリ取り出しを発生させる。条件つきスキップを３バイト分岐と組み合わせると条件つき分岐が作成される。またスキップはある４命令グループ中の残りのバイトを使用することができない状況でも用いることができる。スキップは一つのサイクルで実行され、三つの NOP のグループは三つのサイクルを要する。	The microprocessor 50 fetches instructions in 32-bit chunks called 4-byte instruction groups. These four bytes may contain four 8-bit instructions or some mix of 8-bit and 16-bit or 24-bit instructions. Skip instructions in the microprocessor skip any remaining instructions in a 4-byte instruction group and cause a memory fetch to get the next 4-byte instruction group. Conditional skips when combined with 3-byte branches will create conditional branches. Skips may also be used in situations when no use can be made of the remaining bytes in a 4-instruction group. A skip executes in a single cycle, whereas a group of three NOPs would take three cycles.
SKIP－ALWAYS －この４バイト命令グループ中の残りのいかなる命令もスキップする。プログラムカウンタの最上位の 30 ビットを増分し、次の４バイト命令グループの取り出しに進む。	SKIP-ALWAYS–Skip any remaining instructions in this 4-byte instruction group. Increment the most significant 30 bits of the program counter and proceed to fetch the next 4-byte instruction group.
SKIP－IF－ZERO－パラメータスタックのトップ項目がゼロである場合、４バイト命令グループ中の残りのいかなる命令もスキップ	SKIP-IF-ZERO–If the top item of the parameter stack is zero, skip any remaining instructions in the 4-byte instruction group. Increment

する。プログラムカウンタの最上位の 30 ビットを増分し、次の４バイト命令グループの取り出しに進む。トップ項目がゼロでない場合、次の順次命令を実行する。	the most significant 30 bits of the program counter and proceed to fetch the next 4-byte instruction group. If the top item is not zero, execute the next sequential instruction.
SKIP－IF－POSITIVE－パラメータスタックのトップ項目が“0″に等しい最上位ビット（符号ビット）を有する場合、４バイト命令グループ中の残りのいかなる命令もスキップする。プログラムカウンタの最上位の 30 ビットを増分し、次の４バイト命令グループの取り出しに進む。トップ項目が“0″でない場合、次の順次命令を実行する。	SKIP-IF-POSITIVE–If the top item of the parameter stack has a most significant bit (the sign bit) equal to “0”, skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30 bits of the program counter and proceed to fetch the next 4-byte instruction group. If the top item is not “0” execute the next sequential instruction.
SKIP－IF－NO－CARRY－シフトあるいは算術演算からの桁あげフラグが“1″に等しくない場合、４バイト命令グループ中の残りのいかなる命令もスキップする。プロプログラムカウンタの最上位の 30 ビットを増分し、次の４バイト命令グループの取り出しに進む。桁あげが“1″に等しい場合、次の順次の命令を実行する。	SKIP-IF-NO-CARRY–If the carry flag from a shift or arithmetic operation is not equal to “1”, skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30 bits of the program counter and proceed to fetch the next 4-byte instruction group. If the carry is equal to “1”, execute the next sequential instruction.
SKIP－NEVER－次の順次の命令を実行する（１マシンサイクル遅延する）。	SKIP-NEVER–Execute the next sequential instruction. (Delay 1 machine cycle).
SKIP－IF－NOT－ZERO－パラメータスタックのトップ項目が“0″に等しくない場合、４バイト命令グループ中の残りのいかなる命令もスキップする。プログラムカウンタの最上位の 30 ビットを増分し、次の４バイト命令グループの取り出しに進む。トップ項目が“0″に等しい場合、次の順次命令を実行する。	SKIP-IF-NOT-ZERO–If the top item on the parameter stack is not equal to “0” skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30 bits of the program counter and proceed to fetch the next 4-byte instruction group. If the top item is equal to “0” execute the next sequential instruction.
SKIP－IF－NEGATIVE－パラメータスタックのトップ項目の最上位ビット（符号ビット）が“1″にセットされている場合、４バイト命令グループ中の残りのいかなる命令もスキップする。プログラムカウンタの最上位の 30 ビットを増分し、次の４バイト命令グループの取り出しに進む。トップ項目の最上位ビット（符号ビット）が“0″にセットされている場合、次の順次命令を実行する。	SKIP-IF-NEGATIVE–If the top item on the parameter stack has its most significant bit (sign bit) set to “1”, skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30 bits of the program counter and proceed to fetch the next 4-byte instruction group. If the top item has its most significant bit (sign bit) set to “0” execute the next sequential instruction.
SKIP－IF－CARRY－シフトあるいは算術演算の結果桁あげフラグが“1″にセットされている場合、４バイト命令グループ中の残りのいかなる命令もスキップする。プログラムカウンタの最上位の 30 ビットを増分し、次の４バイト命令グループの取り出しに進む。桁あげフラグが“0″である場合、次の順次命令を実行する。	SKIP-IF-CARRY–If the carry flag is set to “1” as a result of shift or arithmetic operation, skip any remaining instructions in the 4-byte instruction group. Increment the most significant 30 bits of the program counter and proceed to fetch the next 4-byte instruction group. If the carry flag is “0” execute the next sequential instruction.

マイクロループ	Microloops
マイクロループはこのマイクロプロセッサアーキテクチャに特有の機能であり、４バイト命令中での制御されたルーピングを可能とする。マイクロループ命令はループカウンタが“0″であるかどうかをテストし、また別のテストを行うこともできる。ループカウンタが“0″でなく、またテストに合格した場合、４バイト命令グループの第１の命令に対して命令実行が継続され、ループカウンタは減分される。マイクロループ命令は通常４バイト命令グループの最終バイトであるが、これは任意のバイトとすることができる。ループカウンタが“0″である、あるいはテストに合格しない場合、次の命令に対して命令実行が継続される。マイクロループ命令が通常４バイト命令グループの最終バイトである場合、プログラムカウンタの最上位の 30 ビットが増分され、次の４バイト命令グループがメモリから取り出される。“0″に等しいループカウンタ上のループが終了すると、ループカウンタは“0″にとどまる。マイクロループは移動や探索といった短い繰り返し作業をメモリからの取り出し命令を減速することなく行うことを可能にする。	Microloops are a unique feature of the microprocessor architecture which allows controlled looping within a 4-byte instruction group. A microloop instruction tests the loop counter for “0” and may perform an additional test. If the loop counter is not “0” and the test is met, instruction execution continues with the first instruction in the 4-byte instruction group, and the loop counter is decremented. A microloop instruction will usually be the last byte in a 4-byte instruction group, but it can be any byte. If the loop counter is “0” or the test is not met, instruction execution continues with the next instruction. If the microloop instruction is the last byte in the 4-byte instruction group, the most significant 30 bits of the program counter are incremented and the next 4-byte instruction group is fetched from memory. On a termination of the loop on loop counter equal to “0” the loop counter will remain at “0”. Microloops allow short iterative work such as moves and searches to be performed without slowing down to fetch instructions from memory.
例：	Example:
バイト１　FETCH－VIA－Ｘ－AUTOINCREMENT バイト２　STORE－VIA－Ｙ－AUTOINCREMENT バイト３　ULOOP－UNTIL－DONE バイト４　QQQQQQQQ	Byte 1　FETCH-VIA-X-AUTOINCREMENT Byte 2　STORE-VIA-Y-AUTOINCREMENT Byte 3　ULOOP-UNTIL-DONE Byte 4　QQQQQQQQ
この例はブロック移動を行う。転送を開始するにはＸに出所の開始アドレスがロードされる。Ｙには行先の開始アドレスがロードされる。ループカウンタには移動すべき 32 ビットワードの数がロードされる。このマイクロループはそれがゼロに達するまで取り出しと記憶を行いループカウンタをカウントダウンする。QQQQQQQQ はいかなる命令も続きうることを意味する。	This example will perform a block move. To initiate the transfer, X will be loaded with the starting address of the source. Y will be loaded with the starting address of the destination. The loop counter will be loaded with the number of 32-bit words to move. The microloop will fetch and store and count down the loop counter until it reaches zero. QQQQQQQQ indicates any instruction can follow.
マイクロループ命令	Microloop Instructions
ULOOP－UNTIL－DONE－ループカウンタが“0″でない場合、４バイト命令グループ中の第１の命令について実行を継続する。ループ	ULOOP-UNTIL-DONE–If the loop counter is not “0”, continue execution with the first instruction in the 4-byte instruction group.

カウンタを減分する。ループカウンタが“0″である場合、次の命令の実行を継続する。	Decrement the loop counter. If the loop counter is “0” continue execution with the next instruction.
ULOOP－IF－ZERO－ループカウンタが“0″でなく、パラメータスタックのトップ項目が“0″である場合、４バイト命令グループ中の第１の命令について実行を継続する。ループカウンタを減分する。ループカウンタが“0″である場合、あるいはトップ項目が“1″である場合、次の命令の実行を継続する。	ULOOP-IF-ZERO–If the loop counter is not “0” and the top item on the parameter stack is “0”, continue execution with the first instruction in the 4-byte instruction group. Decrement the loop counter. If the loop counter is “0” or the top item is “1”, continue execution with the next instruction.
ULOOP－IF－POSITIVE－ループカウンタが“0″でなく、最上位ビット（符号ビット）が“0″である場合、４バイト命令グループ中の第１の命令について実行を継続する。ループカウンタを減分する。ループカウンタが“0″である場合、あるいはトップ項目が“1″である場合、次の命令の実行を継続する。	ULOOP-IF-POSITIVE–If the loop counter is not “0” and the most significant bit (sign bit) is “0”, continue execution with the first instruction in the 4-byte instruction group. Decrement the loop counter. If the loop counter is “0” or the top item is “1”, continue execution with the next instruction.
ULOOP－IF－NOT－CARRY－CLEAR－ループカウンタが“0″でなく、トップと次で発見された浮動小数点の指数の位置が合っていない場合、４バイト命令グループの第１の命令について実行を継続する。ループカウンタを減分する。ループカウンタが“0″である場合、あるいはこれらの指数の位置が合っている場合、次の命令の実行を継続する。この命令は二つの浮動少数点数を位置合わせするための特殊なシフト命令と組み合わせるように特に設計したものである。	ULOOP-IF-NOT-CARRY-CLEAR–If the loop counter is not “0” and the floating point exponents found in top and next are not aligned, continue execution with the first instruction in the 4-byte instruction group. Decrement the loop counter. If the loop counter is “0” or the exponents are aligned, continue execution with the next instruction. This instruction is specifically designed for combination with special shift instructions to align two floating point numbers.
ULOOP－NEVER－（DECREMENT－LOOP－COUNTER）ループカウンタを減分する。次の命令の実行を継続する。	ULOOP-NEVER–(DECREMENT-LOOP-COUNTER) Decrement the loop counter. Continue execution with the next instruction.
ULOOP－IF－NOT－ZERO－ループカウンタが“0″でなく、パラメータスタックのトップ項目が“0″である場合、４バイト命令グループ中の第１の命令について実行を継続する。ループカウンタを減分する。ループカウンタが“0″である場合、あるいはトップ項目が“1″である場合、次の命令の実行を継続する。	ULOOP-IF-NOT-ZERO–If the loop counter is not “0” and the top item of the parameter stack is “0”, continue execution with the first instruction in the 4-byte instruction group. Decrement the loop counter. If the loop counter is “0” or the top item is “1”, continue execution with the next instruction.
ULOOP－IF－NEGATIVE－ループカウンタが“0″でなく、パラメータスタックのトップ項目の最上位ビット（符号ビット）が“1″である場合、４バイト命令グループ中の第１の命令について実行を継続する。ループカウンタを減分する。ループカウンタが“0″である場合、あるいはパラメータスタックの最上位ビットが“0″である場合、次の命令の実行を継続する。	ULOOP-IF-NEGATIVE–If the loop counter is not “0” and the most significant bit (sign bit) of the top item of the parameter stack is “1”, continue execution with the first instruction in the 4-byte instruction group. Decrement the loop counter. If the loop counter is “0” or the most significant bit of the parameter stack is “0” continue execution with the next instruction.

ULOOP－IF－CARRY－SETーループカウンタが“0″でなく、トップと次で発見された浮動少数点数の指数の位置が合っていない場合、４バイト命令グループの第１の命令について実行を継続する。ループカウンタを減分する。ループカウンタが“0″である場合、あるいはこれらの指数の位置が合っている場合、次の命令の実行を継続する。	ULOOP-IF-CARRY-SET–If the loop counter is not “0” and the exponents of the floating point numbers found in top and next are not aligned, continue execution with the first instruction in the 4-byte instruction group. Decrement the loop counter. If the loop counter is “0” or the exponents are aligned, continue execution with the next instruction.
サブルーチンあるいは割込みからのリターン	Return From Subroutine Or Interrupt
サブルーチンの呼び出しおよび割込み応答によって通常のプログラム実行の再指令が発生する。いずれの場合も、現在のプログラムカウンタがリターンスタック上にプッシュされ、マイクロプロセッサはサブルーチンあるいは割込みサービスルーチンを実行した後、プログラム中の位置に戻ることができる。	Subroutine calls and interrupt acknowledgements cause a redirection of normal program execution. In both cases, the current program counter is pushed onto the return stack, so the microprocessor can return to its place in the program after executing the subroutine or interrupt service routine.
注：サブルーチンの呼び出しおよび割込み応答の際には、プログラムカウンタはすでに増分されており、現在実行中の４バイトグループに続く４バイト命令グループを指示している。命令復号ロジックはマイクロプロセッサが１サイクルでテストを行い、そのテストの結果を条件とするリターンを実行することを可能とする。リターンはリターンスタックからアドレスを取り出し、それをプログラムカウンタに記憶する。	NOTE: When a call to a subroutine or interrupt is acknowledged the program counter has already been incremented and is pointing to the 4-byte instruction group following the 4-byte group currently being executed. The instruction decoding logic allows the microprocessor to perform a test and execute a return conditional on the outcome of the test in a single cycle. A RETURN fetches an address from the return stack and stores it to the program counter.
リターン命令	Return Instructions
RETURN－ALWAYS －リターンスタックからトップ項目を取り出し、それをプログラムカウンタに転送する。	RETURN-ALWAYS–Fetch the top item from the return stack and transfer it to the program counter.
RETURN－IF－ZEROーパラメータスタックのトップ項目が“0″である場合、リターンスタックからトップ項目を取り出し、それをプログラムカウンタに転送する。そうでない場合、次の命令を実行する。	RETURN-IF-ZERO–If the top item on the parameter stack is “0”, fetch the top item from the return stack and transfer it to the program counter. Otherwise execute the next instruction.
RETURN－IF－POSITIVEーパラメータスタックのトップ項目の最上位ビット（符号ビット）が“1″である場合、リターンスタックからトップ項目を取り出し、それをプログラムカウンタに転送する。そうでない場合、次の命令を実行する。	RETURN-IF-POSITIVE–If the most significant bit (sign bit) of the top item on the parameter stack is a “1” fetch the top item from the return stack and transfer it to the program counter. Otherwise execute the next instruction.
RETURN－IF－NOT－CARRY－CLEARートップと次で発見された浮動小数点の指数の位置が合っていない場合、リターンスタックからト	RETURN-IF-NOT-CARRY-CLEAR–If the exponents of the floating point numbers found in top and next are not aligned, fetch the top

ップ項目を取り出し、それをプログラムカウンタに転送する。そうでない場合、次の命令を実行する。	item from the return stack and transfer it to the program counter. Otherwise execute the next instruction.
RETURN－NEVER－次の命令を実行する。	RETURN-NEVER–Execute the next instruction.
(NOP)	(NOP)
RETURN－IF－NOT－ZERO－パラメータスタックのトップ項目が“0″でない場合、リターンスタックからトップ項目を取り出し、それをプログラムカウンタに転送する。そうでない場合、次の命令を実行する。	RETURN-IF-NOT-ZERO–If the top item on the parameter stack is not “0” fetch the top item from the return stack and transfer it to the program counter. Otherwise execute the next instruction.
RETURN－IF－NEGATIVE－パラメータスタックのトップ項目の最上位ビット（符号ビット）が“1″である場合、リターンスタックからトップ項目を取り出し、それをプログラムカウンタに転送する。そうでない場合、次の命令を実行する。	RETURN-IF-NEGATIVE–If the most significant bit (sign bit) of the top item on the parameter stack is a “1” fetch the top item from the return stack and transfer it to the program counter. Otherwise execute the next instruction.
RETURN－IF－CARRY－SET－トップと次で発見された浮動小数点数の指数の位置が合っている場合、リターンスタックからトップ項目を取り出し、それをプログラムカウンタに転送する。そうでない場合、次の命令を実行する。	RETURN-IF-CARRY-SET–If the exponents of the floating point numbers found in top and next are aligned, fetch the top item from the return stack and transfer it to the program counter. Otherwise execute the next instruction.
ダイナミック RAM からのメモリの処理	HANDLING MEMORY FROM DYNAMIC RAM
マイクロプロセッサ 50 は任意の RISC 型アーキテクチャと同様に、チップ上で可能な限り多くの動作を最大速度で処理するよう最適化されている。外部メモリ動作はオンチップメモリの速度が 4nsec から 30nsec であるのに対して 80nsec から 220nsec を要する。外部メモリをアクセスしなければならない時間がある。	The microprocessor 50, like any RISC type architecture, is optimized to handle as many operations as possible on-chip for maximum speed. External memory operations take from 80 nsec to 220 nsec compared with on-chip memory speeds of from 4 nsec to 30 nsec. There is time when external memory must be accessed.
外部メモリは三つのレジスタを用いてアクセスされる。	External memory is accessed using three registers:
Ｘレジスターメモリアクセスに用いることができ、同時に増分あるいは減分することのできる 30 ビットメモリポインタ。	X-Register–A 30-bit memory pointer which can be used for memory access and simultaneously incremented or decremented.
Ｙレジスターメモリアクセスに用いることができ、同時に増分あるいは減分することのできる 30 ビットメモリポインタ。	Y-Register–A 30-bit memory pointer which can be used for memory access and simultaneously incremented or decremented.
プログラムカウンター通常４バイト命令グループを指示するのに用いられる 30 ビットメモリポインタ。外部メモリは PC に対するアドレスでアクセスすることができる。オペランドは他のコンピュータでは“即時”あるいは“直定数”と呼ぶこともある。メモリポインタとして用いられるときは、各動作の後 PC も増分され	Program-Counter–A 30-bit memory pointer normally used to point to 4-byte instruction groups. External memory may be accessed at addresses with respect to the PC. The operands are sometimes called “Immediate” or “Literal” in other computers. When used as memory pointer, the PC is also incremented after each operation.

る。	
メモリロードおよび記憶命令	Memory Load & Store Instructions
FETCH－VIA－Ｘ－Ｘによって指示された 32 ビットメモリの内容を取り出し、それをパラメータスタック上にプッシュする。Ｘは変更されない。	FETCH-VIA-X–Fetch the 32-bit memory content pointed to by X and push it onto the parameter stack. X is unchanged.
FETCH－VIA－Ｙ－Ｙによって指示された 32 ビットメモリの内容を取り出し、それをパラメータスタック上にプッシュする。Ｙは変更されない。	FETCH-VIA-Y–Fetch the 32-bit memory content pointed to by Y and push it onto the parameter stack. Y is unchanged.
FETCH－VIA－Ｘ－AUTOINCREMENT－Ｘによって指示された 32 ビットメモリの内容を取り出し、それをパラメータスタック上にプッシュする。取り出しの後、Ｘの最上位の 30 ビットを増分し、次の 32 ビットワードアドレスを指示する。	FETCH-VIA-X-AUTOINCREMENT–Fetch the 32-bit memory content pointed to by X and push it onto the parameter stack. After fetching, increment the most significant 30 bits of X to point to the next 32-bit word address.
FETCH－VIA－Ｙ－AUTOINCREMENT－Ｙによって指示された 32 ビットメモリの内容を取り出し、それをパラメータスタック上にプッシュする。取り出しの後、Ｙの最上位の 30 ビットを増分し、次の 32 ビットワードアドレスを指示する。	FETCH-VIA-Y-AUTOINCREMENT–Fetch the 32-bit memory content pointed to by Y and push it onto the parameter stack. After fetching, increment the most significant 30 bits of Y to point to the next 32-bit word address.
FETCH－VIA－Ｘ－AUTODECREMENT－Ｘによって指示された 32 ビットメモリの内容を取り出し、それをパラメータスタック上にプッシュする。取り出しの後、Ｘの最上位の 30 ビットを減分し、前の 32 ビットワードアドレスを指示する。	FETCH-VIA-X-AUTODECREMENT–Fetch the 32-bit memory content pointed to by X and push it onto the parameter stack. After fetching, decrement the most significant 30 bits of X to point to the previous 32-bit word address.
FETCH－VIA－Ｙ－AUTODECREMENT－Ｙによって指示された 32 ビットメモリの内容を取り出し、それをパラメータスタック上にプッシュする。取り出しの後、Ｙの最上位の 30 ビットを減分し、前の 32 ビットワードアドレスを指示する。	FETCH-VIA-Y-AUTODECREMENT–Fetch the 32-bit memory content pointed to by Y and push it onto the parameter stack. After fetching, decrement the most significant 30 bits of Y to point to the previous 32-bit word address.
STORE－VIA－Ｘ－パラメータスタックのトップ項目を取り出し、それをＸによって指示された記憶場所に記憶する。Ｘは変更されない。	STORE-VIA-X–Fetch the top item of the parameter stack and store it in the memory location pointed to by X. X is unchanged.
STORE－VIA－Ｙ－パラメータスタックのトップ項目を取り出し、それをＹによって指示された記憶場所に記憶する。Ｙは変更されない。	STORE-VIA-Y–Fetch the top item of the parameter stack and store it in the memory location pointed to by Y. Y is unchanged.
STORE－VIA－Ｘ－AUTOINCREMENT－パラメータスタックのトップ項目を取り出し、それをＸによって指示された記憶場所に記憶す	STORE-VIA-X-AUTOINCREMENT–Fetch the top item of the parameter stack and store it in the memory location pointed to by X. After storing, increment the most significant 30 bits of X to

る。記憶の後、Xの最上位の 30 ビットを増分し、次の 32 ビットワードアドレスを指示する。	point to the next 32-bit word address.
STORE－VIA－Y－AUTOINCREMENT－パラメータスタックのトップ項目を取り出し、それをYによって指示された記憶場所に記憶する。記憶の後、Yの最上位の 30 ビットを増分し、次の 32 ビットワードアドレスを指示する。	STORE-VIA-Y-AUTOINCREMENT–Fetch the top item of the parameter stack and store it in the memory location pointed to by Y. After storing, increment the most significant 30 bits of Y to point to the next 32-bit word address.
STORE－VIA－X－AUTODECREMENT－パラメータスタックのトップ項目を取り出し、それをXによって指示された記憶場所に記憶する。記憶の後、Xの最上位の 30 ビットを減分し、次の 32 ビットワードアドレスを指示する。	STORE-VIA-X-AUTODECREMENT–Fetch the top item of the parameter stack and store it in the memory location pointed to by X. After storing, decrement the most significant 30 bits of X to point to the previous 32-bit word address.
STORE－VIA－Y－AUTODECREMENT－パラメータスタックのトップ項目を取り出し、それをYによって指示された記憶場所に記憶する。記憶の後、Yの最上位の 30 ビットを減分し、次の 32 ビットワードアドレスを指示する。	STORE-VIA-Y-AUTODECREMENT–Fetch the top item of the parameter stack and store it in the memory location pointed to by Y. After storing, decrement the most significant 30 bits of Y to point to the previous 32-bit word address.
FETCH－VIA－PC－プログラムカウンタによって指示された 32 ビットメモリの内容を取り出し、それをパラメータスタック上にプッシュする。取り出しの後、プログラムカウンタの最上位の 30 ビットを増分し、次の 32 ビットワードアドレスを指示する。	FETCH-VIA-PC–Fetch the 32-bit memory content pointed to by the program counter and push it onto the parameter stack. After fetching, increment the most significant 30 bits of the program counter to point to the next 32-bit word address.
＊注：この命令が実行されるとき、PC はこの命令に続く記憶場所を指示している。これには 32 ビット即時オペランドをローディングする効果がある。これは８ビット命令であり、したがって４バイト命令取り出し中に他の８ビット命令と結合される。ひとつの４バイト命令取り出しで一つから四つまでの FETCH－VIA－PC 命令を持つことができる。PC は FETCH－VIA－PC が実行されるたびに増分し、、したがってスタック上に即時オペランドをプッシュすることが可能である。この四つのオペランドはこの命令に続く四つの記憶場所で発見される。	*NOTE: When this instruction executes, the PC is pointing to the memory location following the instruction. The effect is of loading a 32-bit immediate operand. This is an 8-bit instruction and therefore will be combined with other 8-bit instructions in a 4-byte instruction fetch. It is possible to have from one to four FETCH-VIA-PC instructions in a 4-byte instruction fetch. The PC increments after each execution of FETCH-VIA-PC,, so it is possible to push four immediate operands on the stack. The four operands would be the found in the four memory locations following the instruction.
BYTE－FETCH－VIA－X－Xの最上位の 30 ビットによって指示された 32 ビットメモリの内容を取り出す。Xの二つの最下位ビットを用いて 32 ビットメモリ取り出しからの四つのバイトのうちの一つを選択し、そのバイトを 32 ビットフィールドで右寄せし、選択されたバイトを前に先行ゼロを付けてパラメータスタック上にプッシュする。	BYTE-FETCH-VIA-X–Fetch the 32-bit memory content pointed to by the most significant 30 bits of X. Using the two least significant bits of X, select one of four bytes from the 32-bit memory fetch, right justify the byte in a 32-bit field and push the selected byte preceded by leading zeros onto the parameter stack.

BYTE－STORE－VIA－Ｘ－Ｘの最上位の 30 ビットによって指示された 32 ビットメモリの内容を取り出す。パラメータスタックからトップ項目を取り出す。Ｘの二つの最下位ビットを用いて最下位バイトを 32 ビットメモリデータに入れ、その 32 ビットのエンティティＸの最上位の 30 ビットによって指示された場所に書き込む。	BYTE-STORE-VIA-X–Fetch the 32-bit memory content pointed to by the most significant 30 bits of X. Fetch the top item from the parameter stack. Using the two least significant bits of X place the least significant byte into the 32-bit memory data and write the 32-bit entity back to the location pointed to by the most significant 30 bits of X.
メモリアクセス命令の他の効果	Other Effects Of Memory Access Instructions
いかなる取り出し命令もある値をパラメータスタック 74 上にプッシュする。オンチップスタックがいっぱいである場合、このスタックはオフチップメモリスタックにオーバーフローし、その結果メモリサイクルが増える。いかなる記憶命令もパラメータスタック 74 からある値を取り出す。オンチップスタックが空である場合、オフチップメモリスタックから値を取り出すためのメモリサイクルが生成される。	Any fetch instruction will push a value on the parameter stack 74. If the on-chip stack is full, the stack will overflow into off-chip memory stack resulting in an additional memory cycle. Any store instruction will fetch a value from the parameter stack 74. If the on-chip stack is empty, a memory cycle will be generated to fetch a value from off-chip memory stack.
オンチップ変数の処理	Handling On-Chip Variables
高レベル言語はローカル変数の作成の可能であるものが多い。これらの変数は特定の作業手順に用いられ放棄される。ネスティングされた作業手順の場合には、これらの変数の層を維持しなければならない。オンチップ記憶装置はオフチップ RAM より最大で５倍高速であり、したがってローカル変数をチップ上に維持する手段があれば動作をより高速にすることができる。マイクロプロセッサ 50 はローカル変数のオンチップ記憶、およびリターンスタックを介した複数レベルの変数のネスティングの両方の能力を提供する。	High-level languages often allow the creation of local variables. These variables are used by a particular procedure and discarded. In cases of nested procedures, layers of these variables must be maintained. On-chip storage is up to five times faster than off-chip RAM, so a means of keeping local variables on-chip can make operations run faster. The microprocessor 50 provides the capability for both on-chip storage of local variables and nesting of multiple levels of variables through the return stack.
リターンスタック 134 は 16 のオンチップ RAM 位置として実施される。リターンスタック 134 の最も一般的な用途はサブルーチンおよび割込み呼び出しからのリターンアドレスを記憶することである。このマイクロプロセッサはこれらの 16 の位置をアドレス指定可能なレジスタとして用いることを可能にする。これえらの 16 の位置は０－15 からのリターンスタック関連アドレスを示す二つの命令によって読み出しおよび書き込みすることができる。高レベル作業手順がネスティングされると、現在の作業手順変数が前	The return stack 134 is implemented as 16 on-chip RAM locations. The most common use for the return stack 134 is storage of return addresses from subroutines and interrupt calls. The microprocessor allows these 16 locations to also be used as addressable registers. The 16 locations may be read and written by two instructions which indicate a return stack-related address from 0-15. When high-level procedures are nested, the current procedure variables push the previous procedure variables further down the return stack 134. Eventually, the return stack will automatically overflow into off-chip RAM.

の作業手順変数をさらにリターンスタック 134 にプッシュする。最後に、リターンスタックはオフチップ RAM に自動的にオーバーフローする。	
オンチップ変数命令	On-Chip Variable Instructions
READ－LOCAL－VARIABLE XXXX－リターンスタックのトップに対して XXXX 番目の位置を読み出す（XXXX は 0000－1111 の２値の数である）。この項目読み出しをパラメータスタックにプッシュする。	READ-LOCAL-VARIABLE XXXX–Read the XXXXth location with respect to the top of the return stack. (XXXX is a binary number from 0000-1111). Push the item read onto the parameter stack.
他の効果：パラメータスタックがいっぱいである場合、このプッシュ動作はスタックの１項目が外部 RAM に自動的に記憶されるとき一つのメモリサイクルを発生させる。この位置を選択するロジックはモジュロ 16 減算を行う。四つのローカル変数がすでにリターンスタック上にプッシュされており、ある命令が５番目の項目を読み出そうとする場合、未知のデータが返される。	Other Effects: If the parameter stack is full, the push operation will cause a memory cycle to be generated as one item of the stack is automatically stored to external RAM. The logic which selects the location performs a modulo 16 subtraction. If four local variables have been pushed onto the return stack, and an instruction attempts to read the fifth item, unknown data will be returned.
WRITE－LOCAL－VARIABLE XXXX－リターンスタックのトップ項目を取り出し、それをリターンスタックのトップに対して XXXX 番目の位置に書き込む。（XXXX は 0000－1111 の２値の数である）。	WRITE-LOCAL-VARIABLE XXXX–Fetch the top item of the parameter stack and write it into the XXXXth location with respect to the top of the return stack. (XXXX is a binary number from 0000-1111.)
他の効果：パラメータスタックが空である場合、このポップ動作は外部 RAM からパラメータスタック項目を取り出すためのメモリサイクルを発生させる。この位置を選択するロジックはモジュロ 16 減算を行う。四つのローカル変数がすでにリターンスタック上にプッシュされており、ある命令が５番目の項目への書き込みを行おうとする場合、リターンアドレスを破壊する、あるいは他の大きな破壊が発生する可能性がある。	Other Effects: If the parameter stack is empty, the pop operation will cause a memory cycle to be generated to fetch the parameter stack item from external RAM. The logic which selects the location performs a modulo 16 subtraction. If four local variables have been pushed onto the return stack, and an instruction attempts to write to the fifth item, it is possible to destroy return addresses or cause other destruction.
レジスタおよびフリップフロップ転送およびプッシュ命令	Register and Flip-Flop Transfer And Push Instructions
DROP－パラメータスタックからトップ項目を取り出しそれを放棄する。	DROP–Fetch the top item from the parameter stack and discard it.
SWAP－トップパラメータ位置のデータをネクストパラメータスタックのデータと交換する。	SWAP–Exchange the data in the top parameter stack location with the data in the next parameter stack location.
DUP－パラメータスタックのトップ項目を複製し、それをパラメータスタック上にプッシュする。	DUP–Duplicate the top item on the parameter stack and push it onto the parameter stack.

PUSH－LOOP－COUNTER－ループカウンタ中の値をパラメータスタック上にプッシュする。	PUSH-LOOP-COUNTER–Push the value in loop counter onto the parameter stack.
POP－RSTACK－PUSH－TO－STACK－リターンスタックからトップ項目を取り出し、それをパラメータスタック上にプッシュする。	POP-RSTACK-PUSH-TO-STACK–Fetch the top item from the return stack and push it onto the parameter stack.
PUSH－Ｘ－REG－Ｘレジスタ中の値をパラメータスタック上にプッシュする。	PUSH-X-REG–Push the value in the X register onto the parameter stack.
PUSH－STACK－POINTER－パラメータスタックの値をパラメータスタック上にプッシュする。	PUSH-STACK-POINTER–Push the value of the parameter stack pointer onto the parameter stack.
PUSH－RSTACK－POINTER－リターンスタックの値をリターンスタック上にプッシュする。	PUSH-RSTACK-POINTER–Push the value of the return stack pointer onto the return stack.
PUSH－MODE－BITS－モードレジスタの値をパラメータスタック上にプッシュする。	PUSH-MODE-BITS–Push the value of the mode register onto the parameter stack.
PUSH－INPUT－10 の専用入力ビットを読み出し、その値（右寄せされ先行ゼロが加えられる）をパラメータスタック上にプッシュする。	PUSH-INPUT–Read the 10 dedicated input bits and push the value (right justified and padded with leading zeros) onto the parameter stack.
SET－LOOP－COUNTER－パラメータスタックからトップ項目を取り出し、それをループカウンタに記憶する。	SET-LOOP-COUNTER–Fetch the top value from the parameter stack and store it into the loop counter.
POP－STACK－PUSH－TO－RSTACK－パラメータスタックからトップ項目を取り出し、それをリターンスタックにプッシュする。	POP-STACK-PUSH-TO-RSTACK–Fetch the top item from the parameter stack and push it onto the return stack.
SET－Ｘ－REG－パラメータスタックからトップ項目を取り出し、それをＸレジスタに記憶する。	SET-X-REG–Fetch the top item from the parameter stack and store it into the X register.
SET－STACK－POINTER－パラメータスタックからトップ項目を取り出し、それをスタックポインタに記憶する。	SET-STACK-POINTER–Fetch the top item from the parameter stack and store it into the stack pointer.
SET－RSTACK－POINTER－パラメータスタックからトップ項目を取り出し、それをリターンスタックポインタに記憶する。	SET-RSTACK-POINTER–Fetch the top item from the parameter stack and store it into the return stack pointer.
SET－MODE－BITS－パラメータスタックからトップ値を取り出し、それをモードビットに記憶する。	SET-MODE-BITS–Fetch the top value from the parameter stack and store it into the MODE BITS.
SET－OUTPUT－パラメータスタックからトップ項目を取り出し、それを 10 の専用出力ビットに出力する。	SET-OUTPUT–Fetch the top item from the parameter stack and output it to the 10 dedicated output bits.
他の効果：パラメータスタックあるいはリターンスタックをプッシュする、あるいは取り出す命令はスタックがオンチップメモリとオフチップメモリの間でオーバーフローするときメモリサイクルを発生させることがある。	Other Effects: Instructions which push or fetch the parameter stack or return stack may cause a memory cycle as the stacks overflow back and forth between on-chip and off-chip memory.

短い直定数のローディング	Loading A Short Literal
レジスタ転送命令の特殊なケースが８ビット直定数をパラメータスタック上にプッシュするのに用いられる。この命令はこのプッシュすべき８ビットが４バイト命令グループの最終バイトにあることを必要とする。直定数をロードするこの命令の演算コードは命令グループ中の他の三つのバイトのうちのいずれにあってもよい。	A special case of register transfer instruction is used to push an 8-bit literal onto the parameter stack. This instruction requires that the 8 bits to be pushed reside in the last byte of a 4-byte instruction group. The instruction op-code loading the literal may reside in any of the other three bytes in the instruction group.
例：	Example:
バイト１　　バイト２　　バイト３ LOAD-SHORT-LITERAL　　QQQQQQQQ　　QQQQQQQQ バイト４ 00001111	BYTE 1　　BYTE 2　　BYTE 3 LOAD-SHORT-LITERAL　QQQQQQQQ　QQQQQQQQ BYTE 4 00001111
この例では QQQQQQQQ は任意の他の８ビット命令を示す。バイト１が実行されるときバイト４からの２値 00001111（16 進数の 0f）がパラメータスタック上にプッシュされる（右寄せされ先行ゼロが付けられる）。次にバイト２とバイト３の命令が実行される。マイクロプロセッサの命令復号器はバイト４を実行しないことを知っている。三つの同じ８ビットの値を次のようにプッシュすることが可能である。	In this example, QQQQQQQQ indicates any other 8-bit instruction. When Byte 1 is executed, binary 00001111 (hexadecimal 0f) from Byte 4 will be pushed (right justified and padded by leading zeros) onto the parameter stack. Then the instructions in Byte 2 and Byte 3 will execute. The microprocessor instruction decoder knows not to execute Byte 4. It is possible to push three identical 8-bit values as follows:
バイト１　　バイト２ LOAD-SHORT-LITERAL　　LOAD-SHORT-LITERAL バイト３　　バイト４ LOAD-SHORT-LITERAL　　00001111	BYTE 1　　BYTE 2 LOAD-SHORT-LITERAL　　LOAD-SHORT-LITERAL BYTE 3　　BYTE 4 LOAD-SHORT-LITERAL　　00001111
SHORT－LITERAL－INSTRUCTION	SHORT-LITERAL-INSTRUCTION
SHORT－LITERAL－INSTRUCTIONLOAD－SHORT－LITERAL－現在の４バイト命令グループのバイト４で発見した８ビットの値をパラメータスタック上にプッシュする。	LOAD-SHORT-LITERAL–Push the 8-bit value found in Byte 4 of the current 4-byte instruction group onto the parameter stack.
ロジック命令	Logic Instructions
論理演算および数理演算ではスタックを一つあるいは二つのオ	Logic operations and math operations used the stack for the source

ペランドの出所用に、また結果の行先として用いた。スタックの構成は式の評価に特に便利な構成になっている。トップはパラメータスタック 74 のトップの値を示す。ネクストはパラメータスタック 74 のトップの値の次の値を示す。	of one or two operands and as the destination for results. The stack organization is a particularly convenient arrangement for evaluating expressions. Top indicates the top value on the parameter stack 74. Next indicates the next to top value on the parameter stack 74.
AND－パラメータスタックからトップとネクストを取り出し、これらの二つのオペランドに論理積演算を行いその結果をパラメータスタック上にプッシュする。	AND–Fetch top and next from the parameter stack, perform the logical AND operation on these two operands, and push the result onto the parameter stack.
OR－パラメータスタックからトップとネクストを取り出し、これらの二つのオペランドに論理和演算を行い、その結果をパラメータスタック上にプッシュする。	OR–Fetch top and next from the parameter stack, perform the logical OR operation on these two operands, and push the result onto the parameter stack.
XOR－パラメータスタックからトップとネクストを取り出し、これらの二つのオペランドに排他的論理和演算を行い、その結果をパラメータスタック上にプッシュする。	XOR–Fetch top and next from the parameter stack, perform the logical exclusive OR on these two operands, and push the result onto the parameter stack.
BIT－CLEAR－パラメータスタックからトップとネクストを取り出し、ネクストのすべてのビットをトグルし、トップに論理積演算を行い、その結果をパラメータスタック上にプッシュする（この命令を理解する別の方法としては、これをネクストにセットされたトップの中のすべてのビットをクリアするものと考えることである）。	BIT-CLEAR–Fetch top and next from the parameter stack, toggle all bits in next, perform the logical AND operation on top, and push the result onto the parameter stack. (Another way of understanding this instruction is thinking of it as clearing all bits in top that are set in next.)
数理命令	Math Instructions
数理命令はオペランドとして用いるためにパラメータスタック 74 のトップ項目とトップの次の項目を取り出す。その結果はパラメータスタックにプッシュされる。桁あげフラグが ALU 結果の“33 番目のビット”をラッチするのに用いられる。	Math instructions fetch the top item and next to top item of the parameter stack 74 to use as the operands. The results are pushed back on the parameter stack. The carry flag is used to latch the “33rd bit” of the ALU result.
ADD－パラメータスタックからトップ項目とトップの次の項目を取り出し、その値を加算し、その結果をパラメータスタックにプッシュする。桁あげフラグを変更するこもできる。	ADD–Fetch the top item and next to top item from the parameter stack, add the values together and push the result back on the parameter stack. The carry flag may be changed.
ADD－WITH－CARRY－パラメータスタックからトップ項目とトップの次の項目を取り出し、その値を加算する。桁あげフラグが“1″である場合、その結果を増分する。最終結果をパラメータスタックにプッシュする。桁あげフラグを変更することもできる。	ADD-WITH-CARRY–Fetch the top item and the next to top item from the parameter stack, add the values together. If the carry flag is “1” increment the result. Push the ultimate result back on the parameter stack. The carry flag may be changed.
ADD－X－パラメータスタックからトップ項目を取り出し、パラメ	ADD-X–Fetch the top item from the parameter stack and read the third item from the top of the parameter stack. Add the values together

ータスタックのトップから三番目の項目を読み出す。その値を加算し、その結果をパラメータスタックにプッシュする。桁あげフラグを変更することもできる。	and push the result back on the parameter stack. The carry flag may be changed.
SUB－パラメータスタックからトップ項目とトップの次の項目を取り出す。トップからネクストを引き、その結果をパラメータスタックにプッシュする。桁あげフラグを変更することもできる。	SUB–Fetch the top item and next to top item from the parameter stack, Subtract next from top and push the result back on the parameter stack. The carry flag may be changed.
SUB－WITH－CARRY－パラメータスタックからトップ項目とトップの次の項目を取り出す。トップからネクストを引く。桁あげフラグが“1″である場合、その結果を増分する。最終結果をパラメータスタックにプッシュする。桁あげフラグを変更することもできる。	SUB-WITH-CARRY–Fetch the top item and next to top item from the parameter stack. Subtract next from top. If the carry flag is “1” increment the result. Push the ultimate result back on the parameter stack. The carry flag may be changed.
SUB－X	SUB-X–
SIGNED－MULT－STEP－	SIGNED-MULT-STEP–
UNSIGNED－MULT－STEP－	UNSIGNED-MULT-STEP–
SIGNED－FAST－MULT－	SIGNED-FAST-MULT–
FAST－MULT－STEP－	FAST-MULT-STEP–
UNSIGNED－DIV－STEP－	UNSIGNED-DIV-STEP–
GENERATE－POLYNOMINAL－	GENERATE-POLYNOMIAL–
ROUND－	ROUND–
COMPARE－パラメータスタックからトップ項目とトップの次の項目を取り出す。トップからネクストを引く。その結果が“0″に等しい最上位ビットを有する（結果は正である）場合、その結果をパラメータスタックにプッシュする。その結果が“1″に等しい最上位ビットを有する（結果は負である）場合、トップの古い値をパラメータスタックにプッシュする。桁あげフラグが影響されることもある。	COMPARE–Fetch the top item and next to top item from the parameter stack. Subtract next from top. If the result has the most significant bit equal to “0” (the result is positive), push the result onto the parameter stack. If the result has the most significant bit equal to “1” (the result is negative), push the old value of top onto the parameter stack. The carry flag may be affected.
SHIFT/ROTATE	SHIFT/ROTATE
SHIFT－LEFT－トップパラメータスタック項目を左に１ビットシフトする。桁あげフラグはトップの最下位ビットにシフトされる。	SHIFT-LEFT–Shift the top parameter stack item left 1 bit. The carry flag is shifted into the least significant bit of top.
SHIFT－RIGHT－トップパラメータスタック項目を右に１ビットシフトする。トップの最下位ビットが桁あげフラグにシフトされる。ゼロがトップの最上位ビットにシフトされる。	SHIFT-RIGHT–Shift the top parameter stack item right 1 bit. The least significant bit of top is shifted into the carry flag. Zero is shifted into the most significant bit of top.

DOUBLE－SHIFT－LEFT－パラメータスタックのトップ項目を 64 ビット数の最上位ワードとして、ネクストスタック項目を最下位ワードとして扱うと、結合された 64 ビットのエンティティは左に 1 ビットシフトされる。桁あげフラグはネクストの最下位ビットにシフトされる。	DOUBLE-SHIFT-LEFT–Treating the top item of the parameter stack as the most significant word of a 64-bit number and the next stack item as the least significant word, shift the combined 64-bit entity left 1 bit. The carry flag is shifted into the least significant bit of next.
DOUBLE－SHIFT－RIGHT－パラメータスタックのトップ項目を 64 ビット数の最上位ワードとして、ネクストスタック項目を最下位ワードとして扱うと、結合された 64 ビットのエンティティは右に 1 ビットシフトされる。ネクストの最下位ビットは桁あげフラグにシフトされる。ゼロはトップの最上位ビットにシフトされる。	DOUBLE-SHIFT-RIGHT–Treating the top item of the parameter stack as the most significant word of a 64-bit number and the next stack item as the least significant word, shift the combined 64-bit entity right 1 bit. The least significant bit of next is shifted into the carry flag. Zero is shifted into the most significant bit of top.
他の命令	Other Instructions
FLUSH－STACK－すべてのオンチップパラメータスタック位置をオフチップ RAM に移す（この命令は多重タスク化アプリケーションに有用である）。この命令はオンチップスタックの深さを保持するカウンタにアクセスし、０から 16 までの外部メモリサイクルを要することがある。	FLUSH-STACK–Empty all on-chip parameter stack locations into off-chip RAM. (This instruction useful for multitasking applications). This instruction accesses a counter which holds the depth of the on-chip stack and can require from 0 to 16 external memory cycles.
FLUSH－RSTACK－すべてのオンチップリターンスタック位置をオフチップ RAM に移す（この命令は多重タスク化アプリケーションに有用である）。この命令はオンチップリターンスタックの深さを保持するカウンタにアクセスし、０から 16 までの外部メモリサイクルを要することがある。	FLUSH-RSTACK–Empty all on-chip return stack locations into off-chip RAM. (This instruction is useful for multitasking applications). This instruction accesses a counter which holds the depth of the on-chip return stack and can require from 0 to 16 external memory cycles.
当該技術に精通するものにはここに図示じ説明したこの発明の態様および細部にさまざまな変更を加えうることは明らかであろう。かかる変更は添付クレームの精神と範囲に含まれるものである。	It should further be apparent to those skilled in the art that various changes in form and details of the invention as shown and described may be made. It is intended that such changes be included within the spirit and scope of the claims appended hereto.

45

フラグはネクストの最下位ビットにシフトされる。

DOUBLE－SHIFT－RIGHT－パラメータスタックのトップ項目を64ビット数の最上位ワードとして、ネクストスタック項目を最下位ワードとして扱うと、結合された64ビットのエンティティは右に１ビットシフトされる。ネクストの最下位ビットは桁あげフラグにシフトされる。ゼロはトップの最上位ビットにシフトされる。

他の命令

FLUSH－STACK－すべてのオンチップパラメータスタック位置をオフチップRAMに移す（この命令は多重タスク化アプリケーションに有用である）。この命令はオンチップスタックの深さを保持するカウンタにアクセスし、０＊

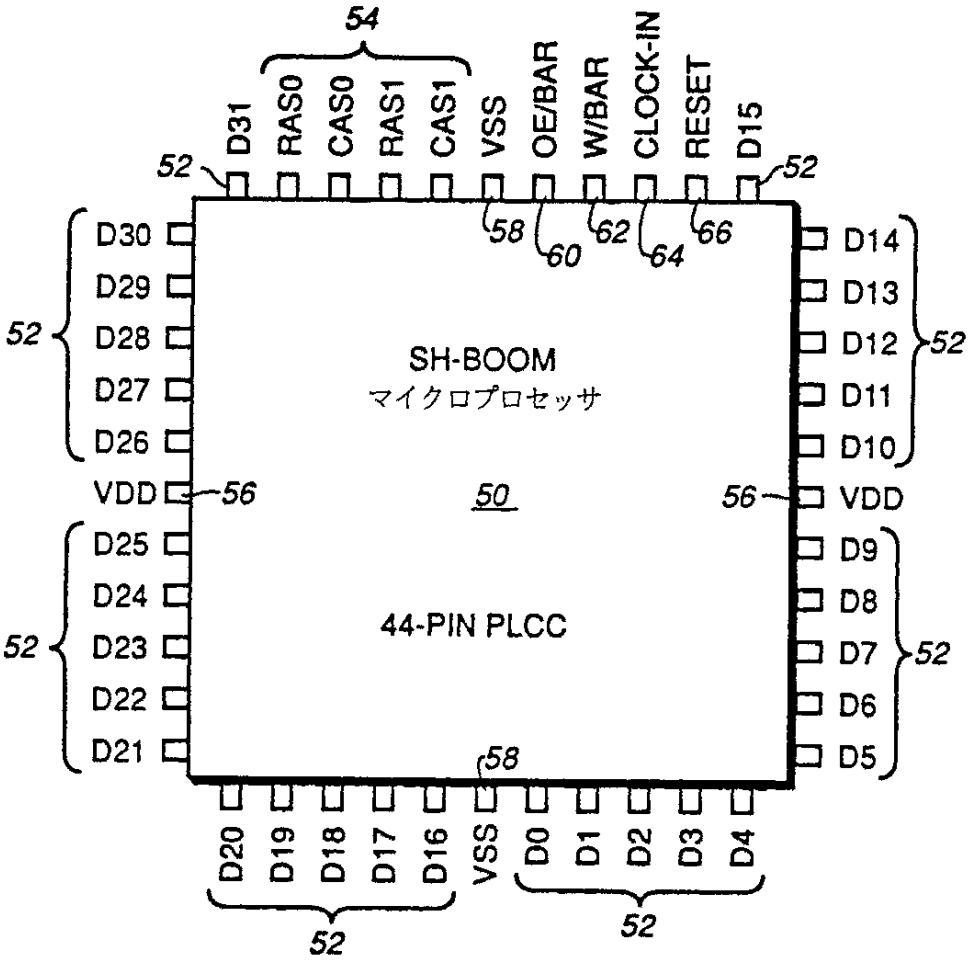
46

＊から16までの外部メモリサイクルを要することがある。

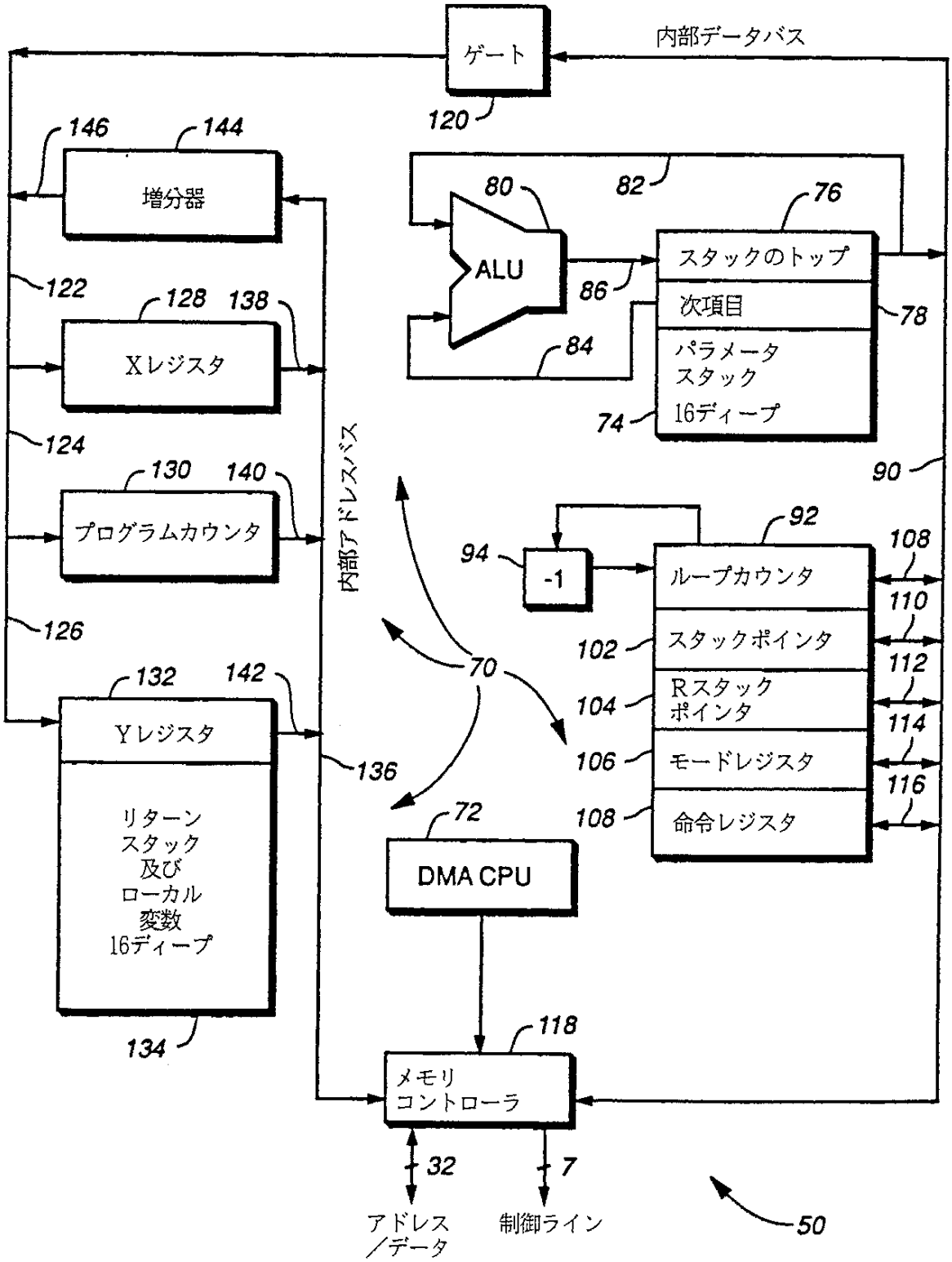
FLUSH－RSTACK－すべてのオンチップリターンスタック位置をオフチップRAMに移す（この命令は多重タスク化アプリケーションに有用である）。この命令はオンチップリターンスタックの深さを保持するカウンタにアクセスし、０から16までの外部メモリサイクルを要することがある。

当該技術に精通するものにはここに図示じ説明したこの発明の態様および細部にさまざまな変更を加えうることは明らかであろう。かかる変更は添付クレームの精神と範囲に含まれるものである。

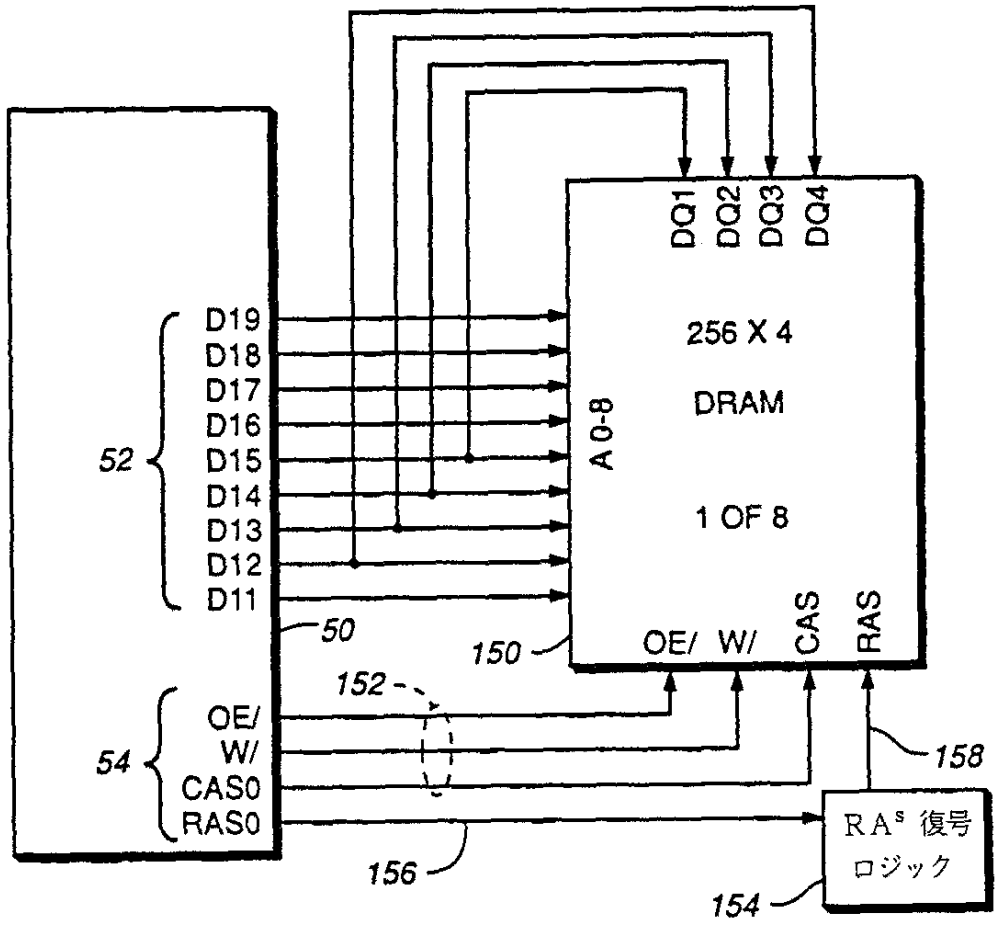
【第１図】 FIG. 1



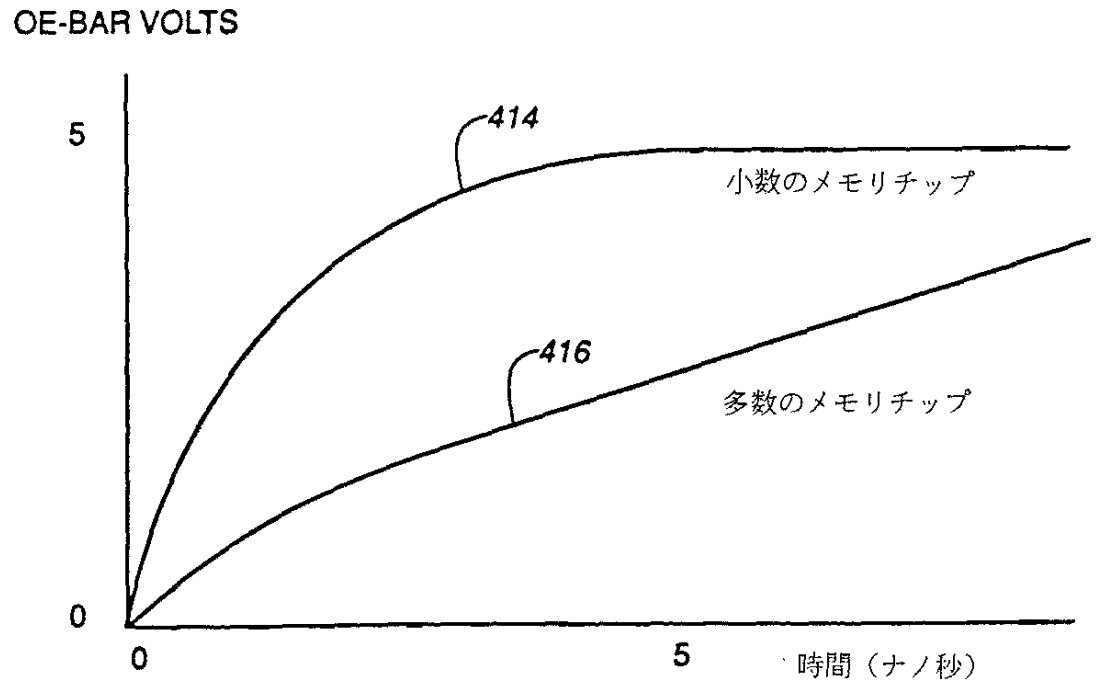
【第２図】 FIG. 2


ゲート
内部データバス
120
146
144
増分器
80
82
76
ALU
スタックのトップ
86
次項目
78
84
パラメータ
スタック
16ディープ
74
122
128
138
Xレジスタ
124
130
140
プログラムカウンタ
内部アドレスバス
90
92
94
-1
ループカウンタ
108
110
102
スタックポインタ
112
104
Rスタック
ポインタ
114
106
モードレジスタ
116
108
命令レジスタ
126
70
132
142
Yレジスタ
136
72
DMA CPU
リターン
スタック
及び
ローカル
変数
16ディープ
134
118
メモリ
コントローラ
32
7
アドレス
／データ
制御ライン
50

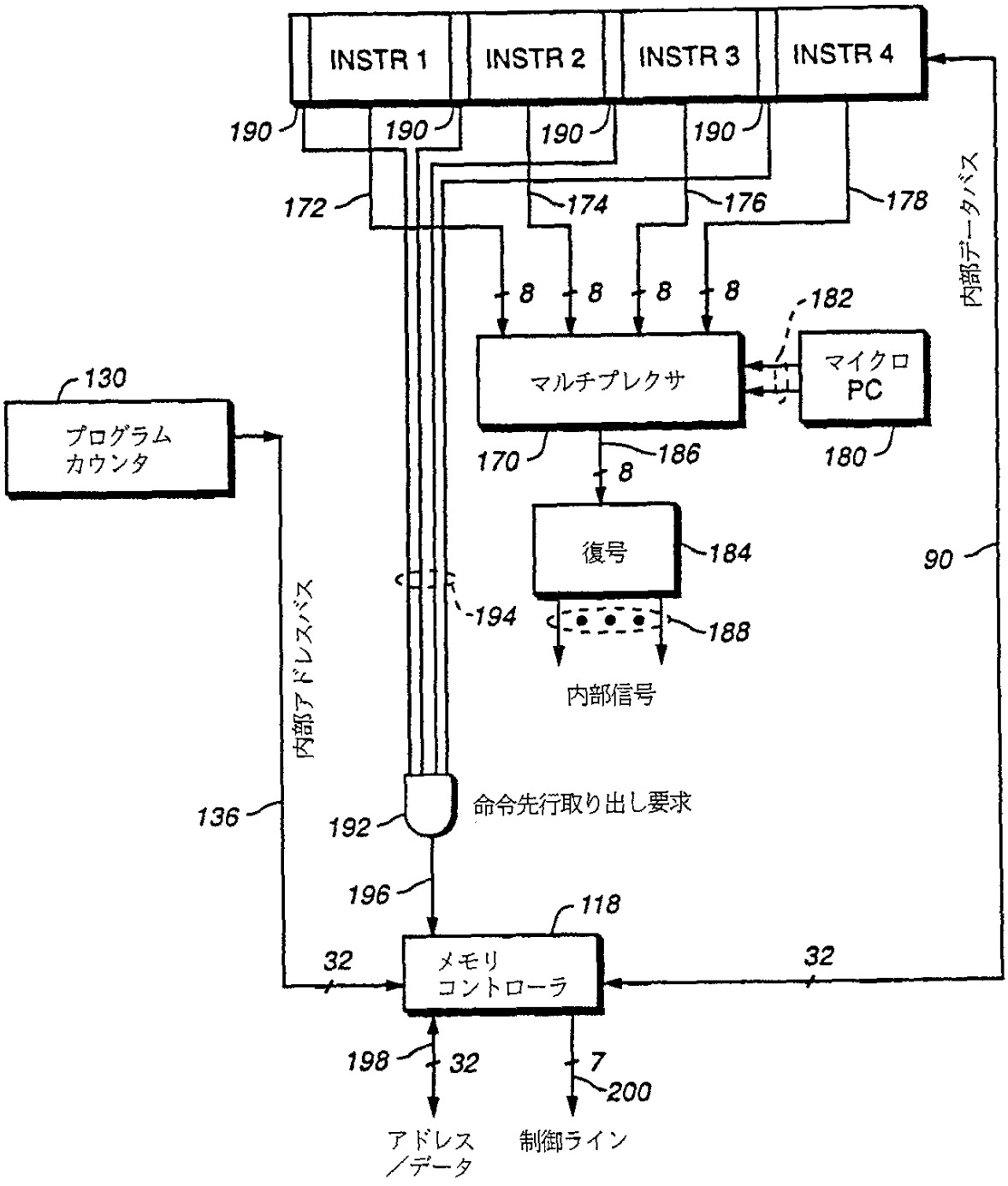
【第３図】 FIG. 3



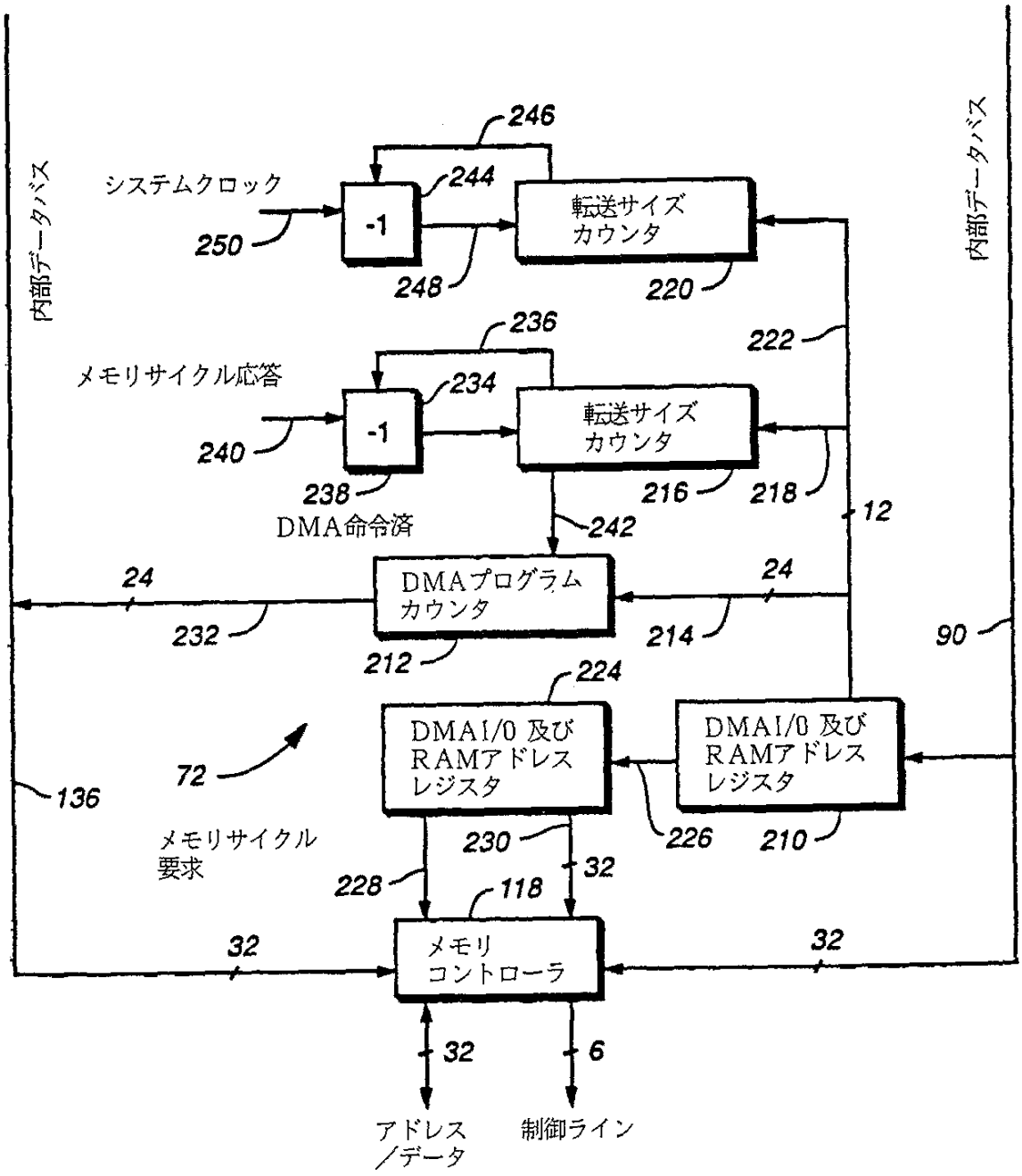
【第１５図】 FIG. 15



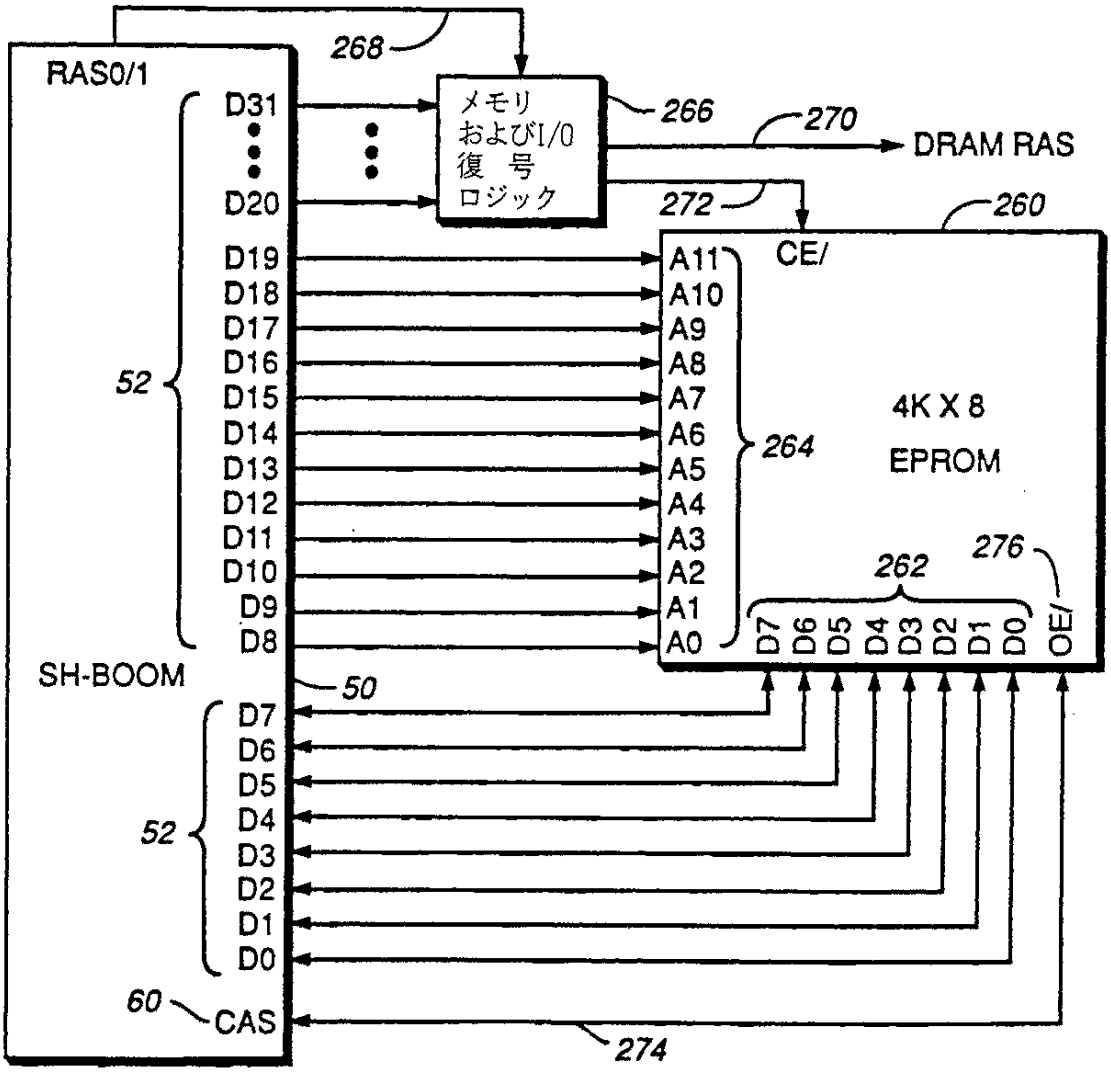
【第４図】FIG. 4


INSTR 1
INSTR 2
INSTR 3
INSTR 4
190
172
174
176
178
8
182
マルチプレクサ
マイクロ PC
130
プログラム カウンタ
170
186
180
復号
184
90
194
188
内部信号
内部データバス
内部アドレスバス
136
192
命令先行取り出し要求
196
118
32
メモリ コントローラ
198
7
200
アドレス ／データ
制御ライン

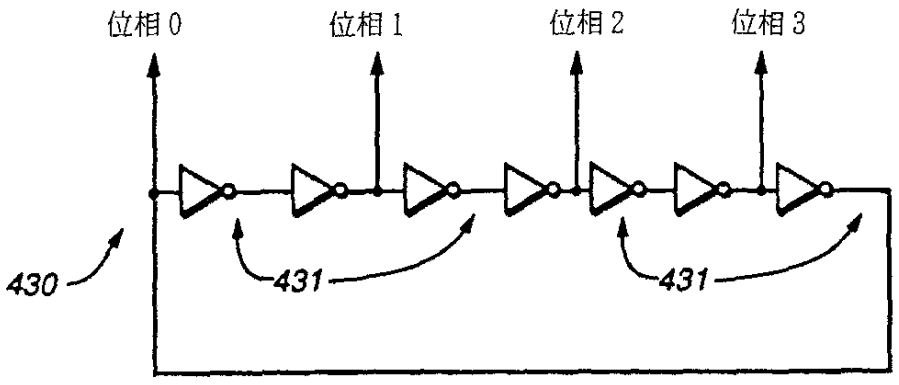
【第５図】 FIG. 5


内部データバス
システムクロック
250
-1
244
246
転送サイズ
カウンタ
220
248
内部データバス
222
メモリサイクル応答
240
-1
234
236
転送サイズ
カウンタ
216
218
238
DMA命令済
242
12
24
DMAプログラム
カウンタ
24
232
214
212
90
224
DMAI/0 及び
RAMアドレス
レジスタ
DMAI/0 及び
RAMアドレス
レジスタ
72
136
メモリサイクル
要求
230
226
210
228
32
118
32
メモリ
コントローラ
32
32
6
アドレス
／データ
制御ライン

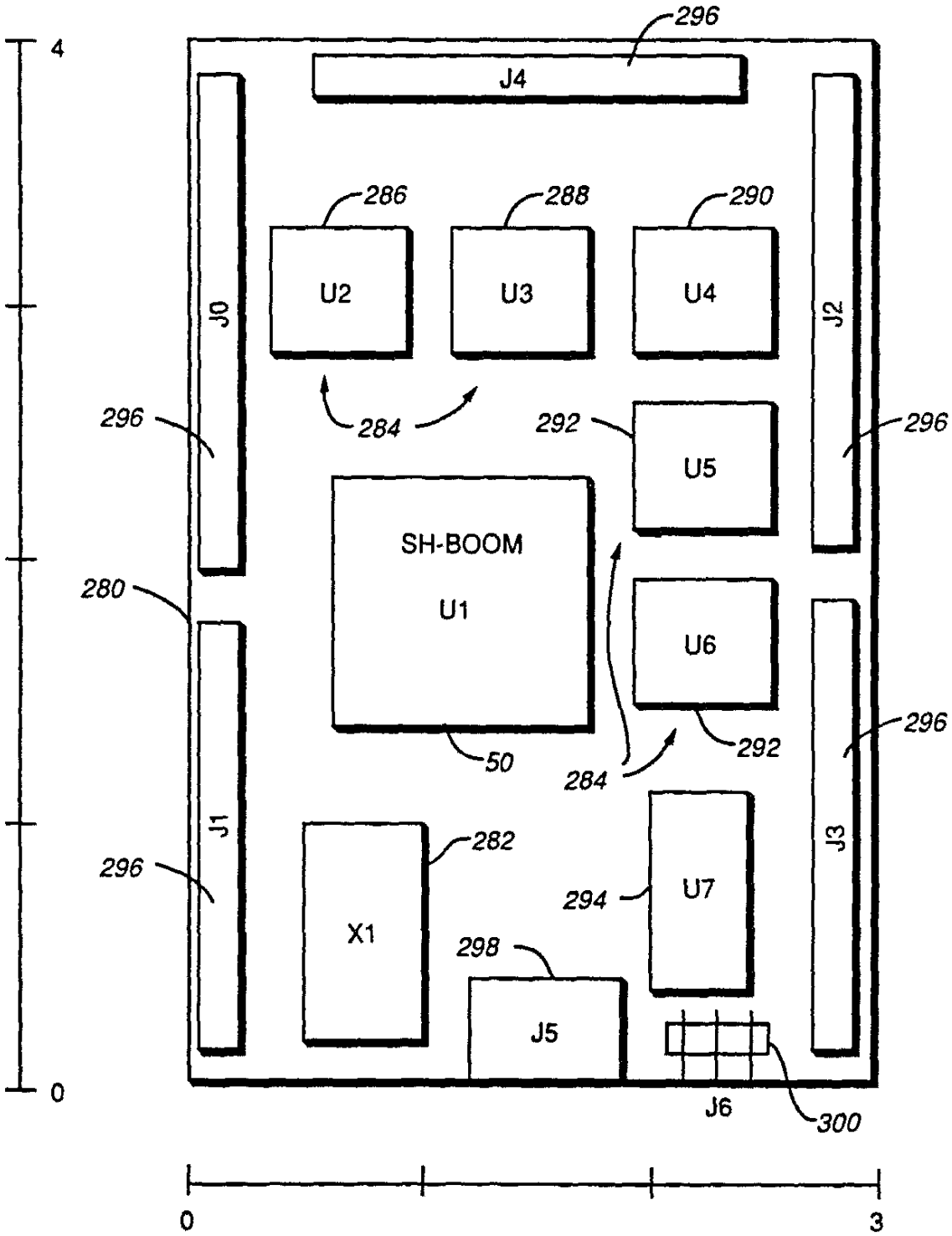
【第６図】FIG. 6



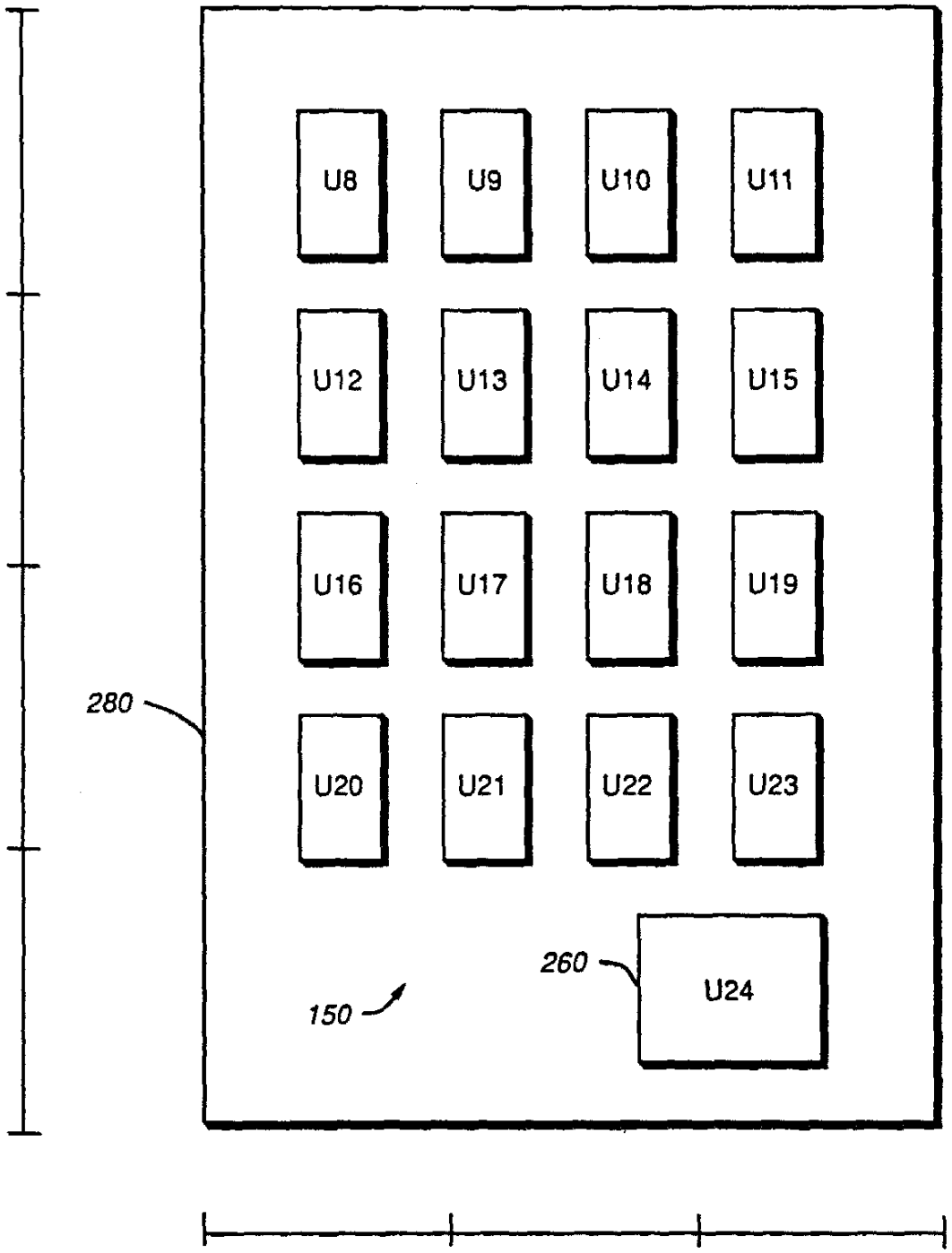
【第１８図】 FIG. 18



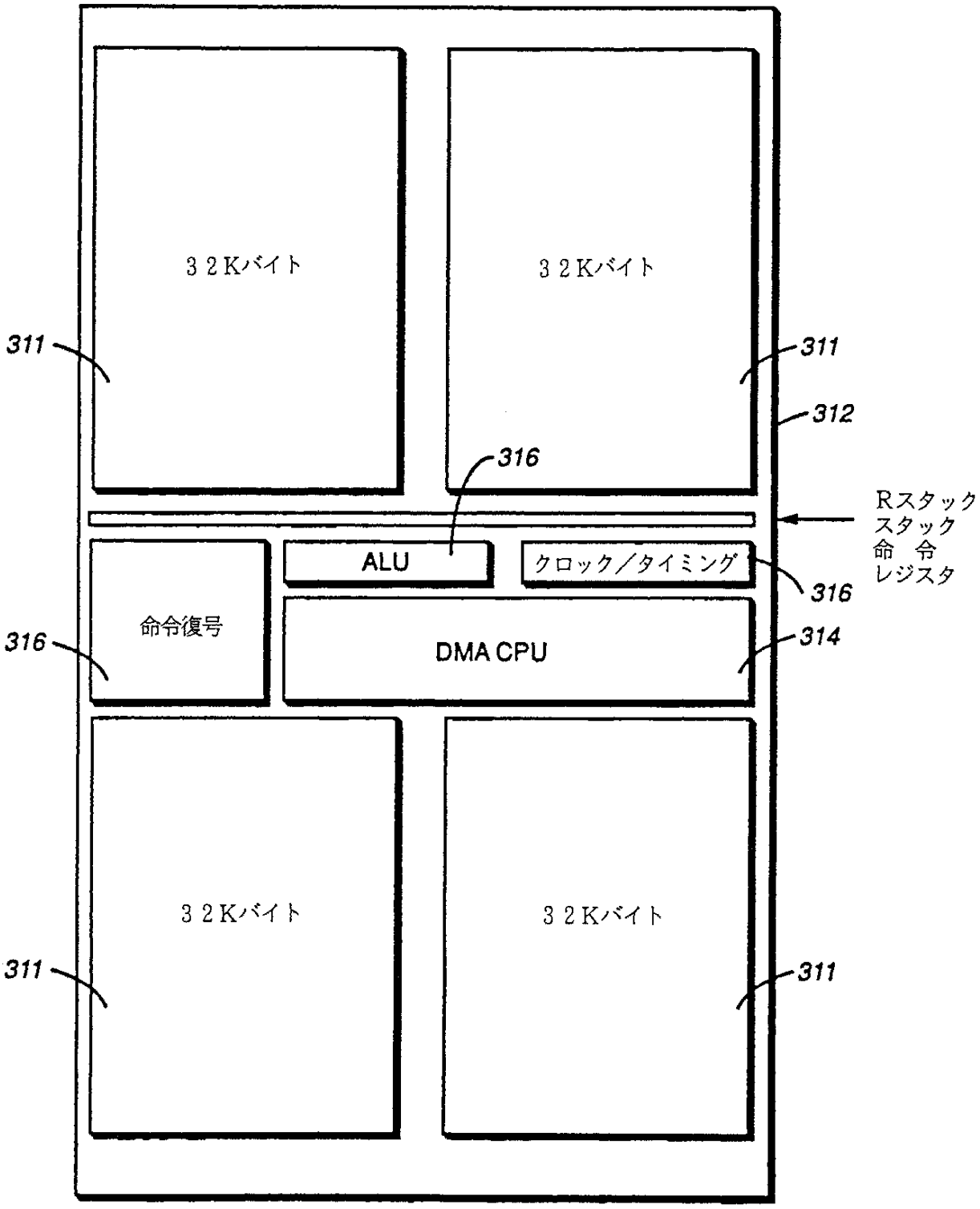
【第７図】 FIG. 7


296
J4
286
288
290
U2
U3
U4
J0
J2
284
296
292
296
U5
SH-BOOM
U1
280
U6
296
50
292
284
J1
J3
282
296
294
U7
X1
298
J5
J6
300
4
0
0
3

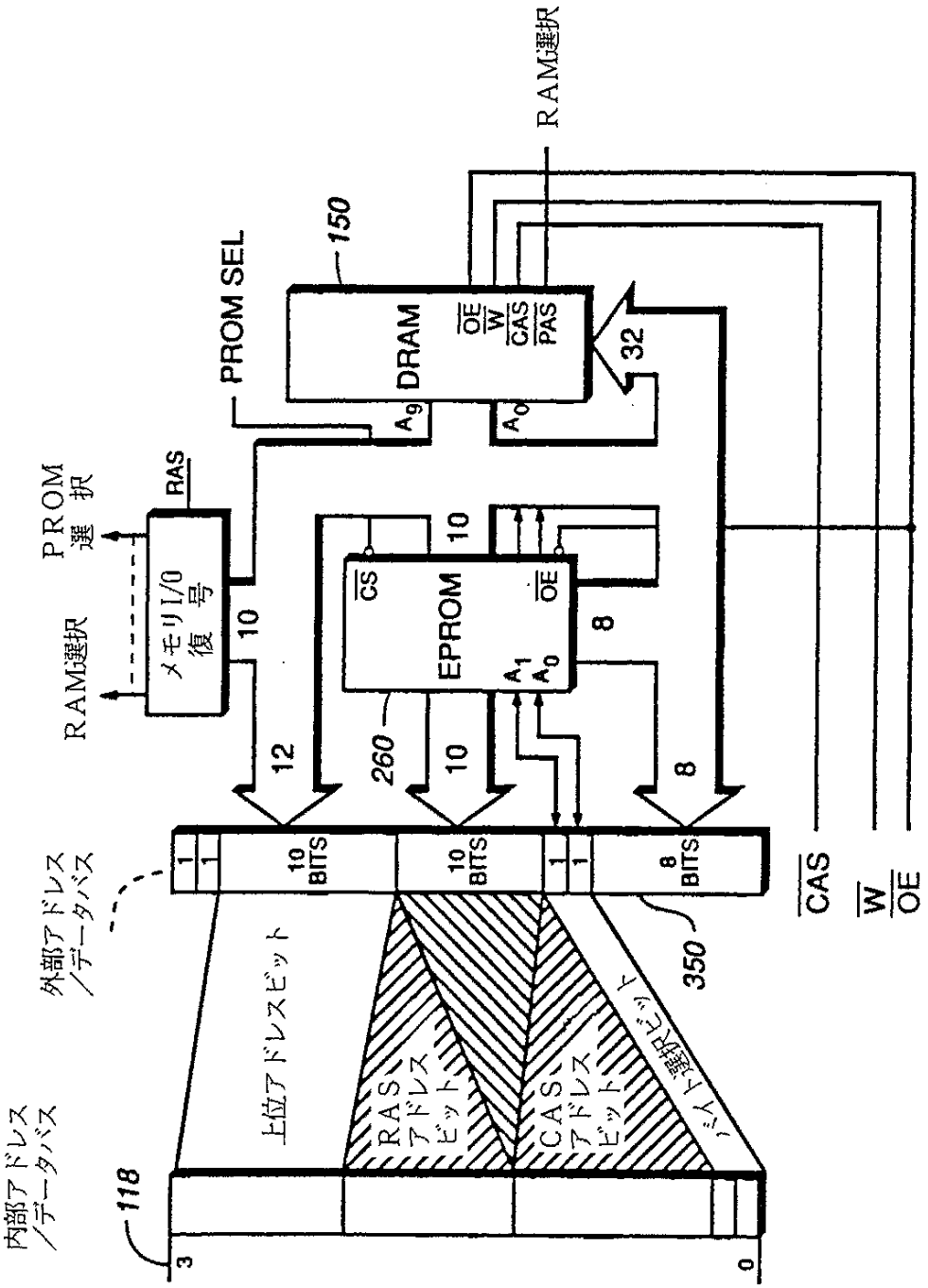
【第８図】 FIG. 8


U8
U9
U10
U11
U12
U13
U14
U15
U16
U17
U18
U19
280
U20
U21
U22
U23
260
U24
150

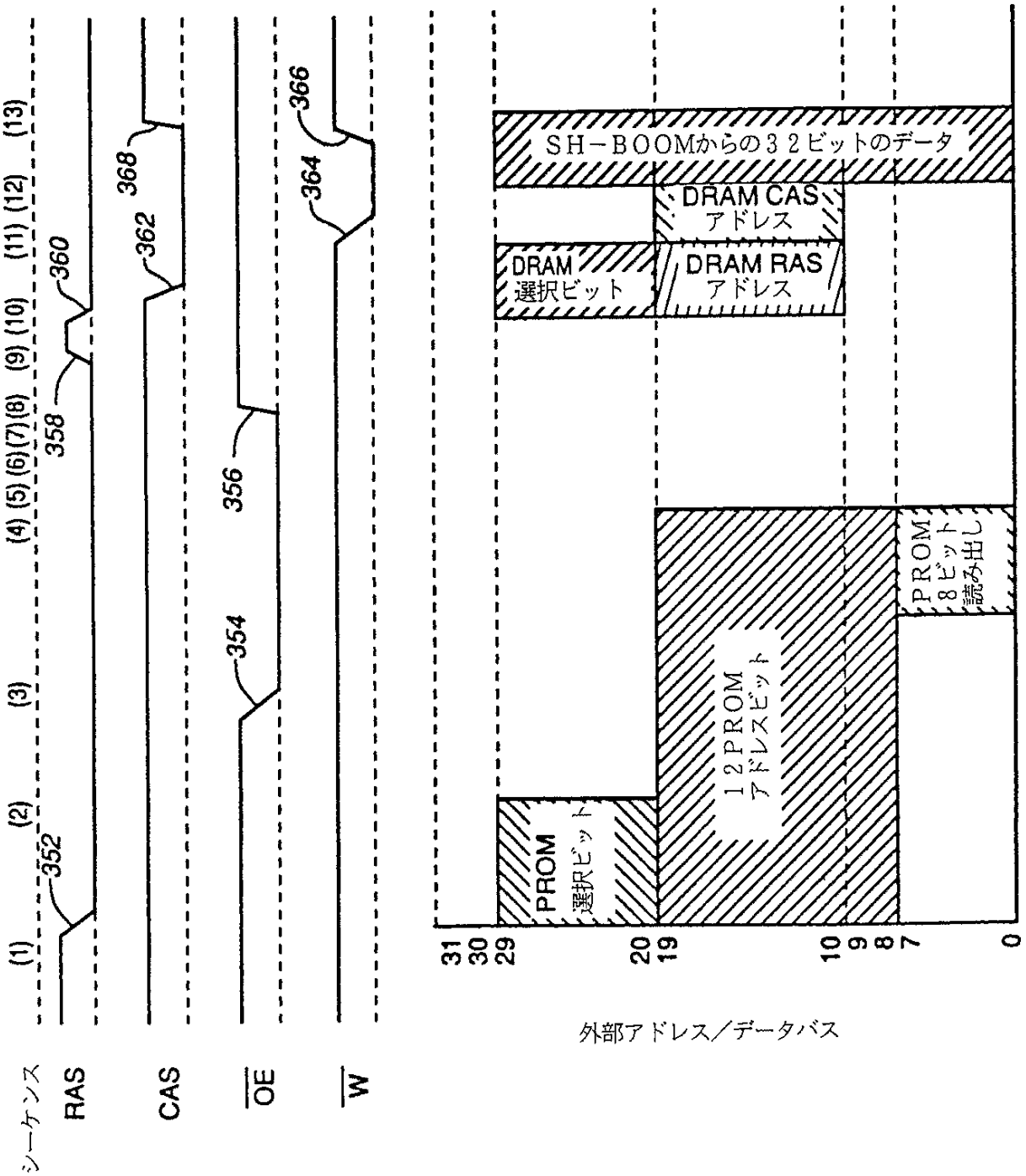
【第９図】 FIG. 9


３２Kバイト
３２Kバイト
311
311
312
316
Rスタック
スタック
命　令
レジスタ
ALU
クロック／タイミング
316
命令復号
316
DMA CPU
314
３２Kバイト
３２Kバイト
311
311

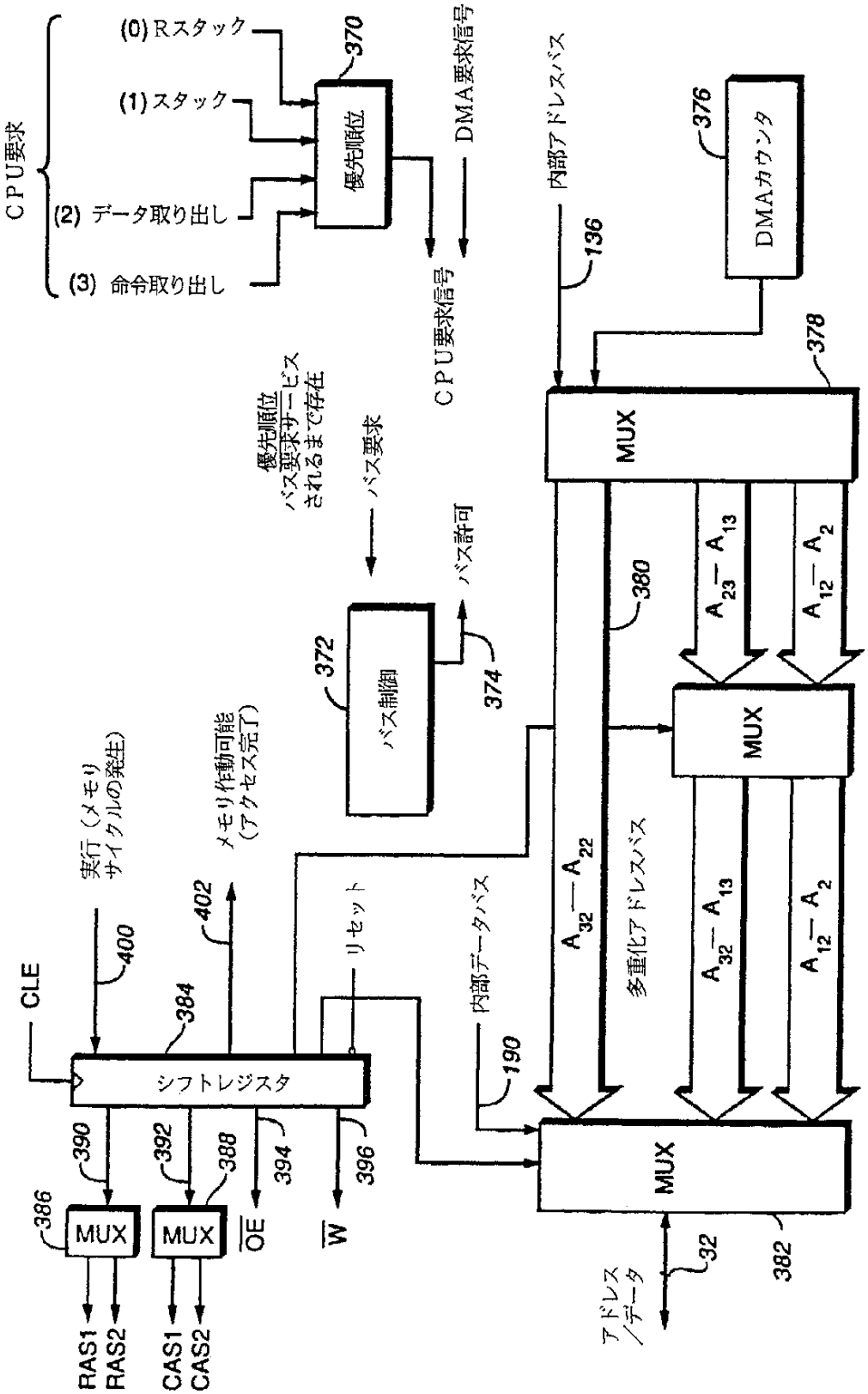
【第１０図】FIG. 10


RAM選択
150
PROM SEL
DRAM
OE
W
CAS
RAS
32
A9
A0
PROM
選択
RAS
メモリI/O
復号
10
CS
EPROM
OE
A1
A0
8
RAM選択
12
260
10
8
外部アドレス
/データバス
1
1
10
BITS
10
BITS
1
1
8
BITS
CAS
W
OE
350
上位アドレスビット
RAS
アドレス
ビット
CAS
アドレス
ビット
バイト選択ビット
内部アドレス
/データバス
118
3
0

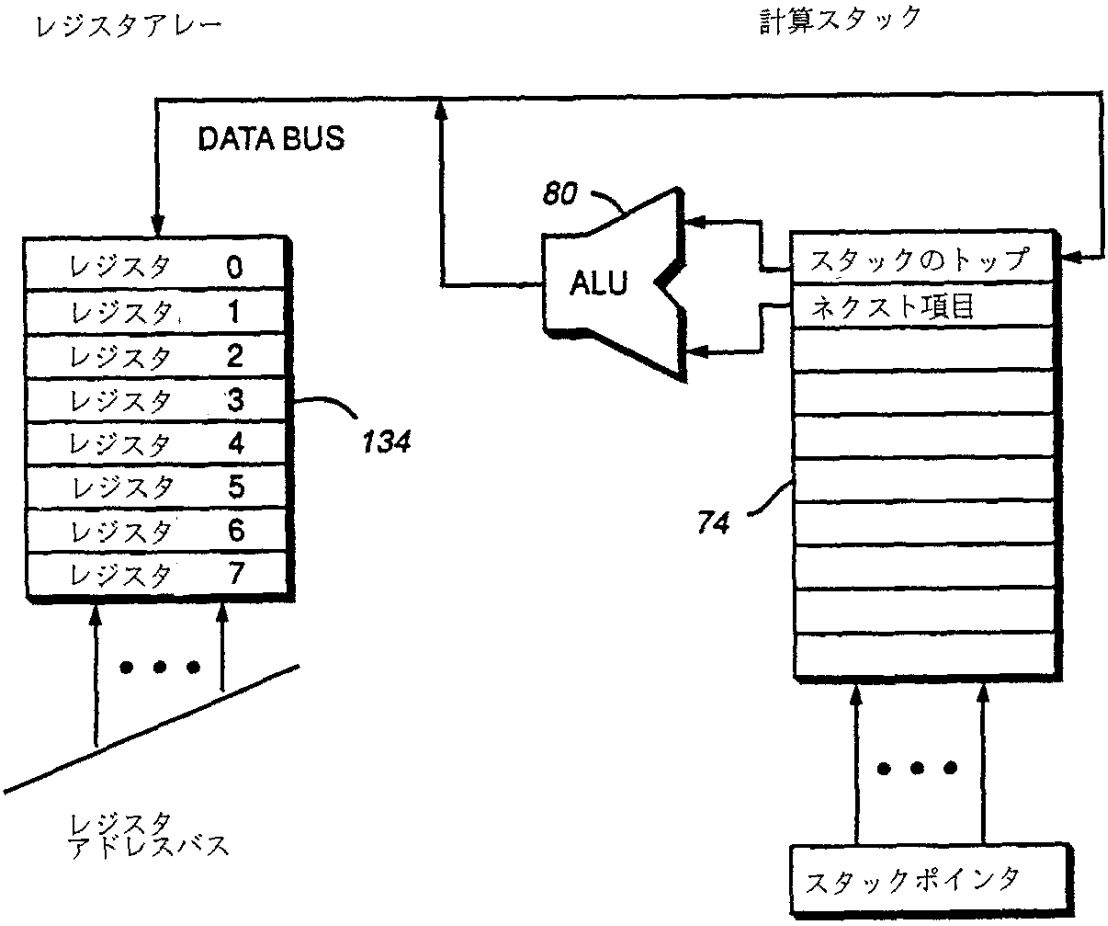
【第１１図】 FIG. 11


シーケンス
RAS
CAS
OE
W
(1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13)
352
354
356
358
360
362
364
366
368
SH－BOOMからの３２ビットのデータ
DRAM CAS アドレス
DRAM 選択ビット
DRAM RAS アドレス
PROM 8ビット読み出し
12PROM アドレスビット
PROM 選択ビット
31 30 29 20 19 10 9 8 7 0
外部アドレス／データバス

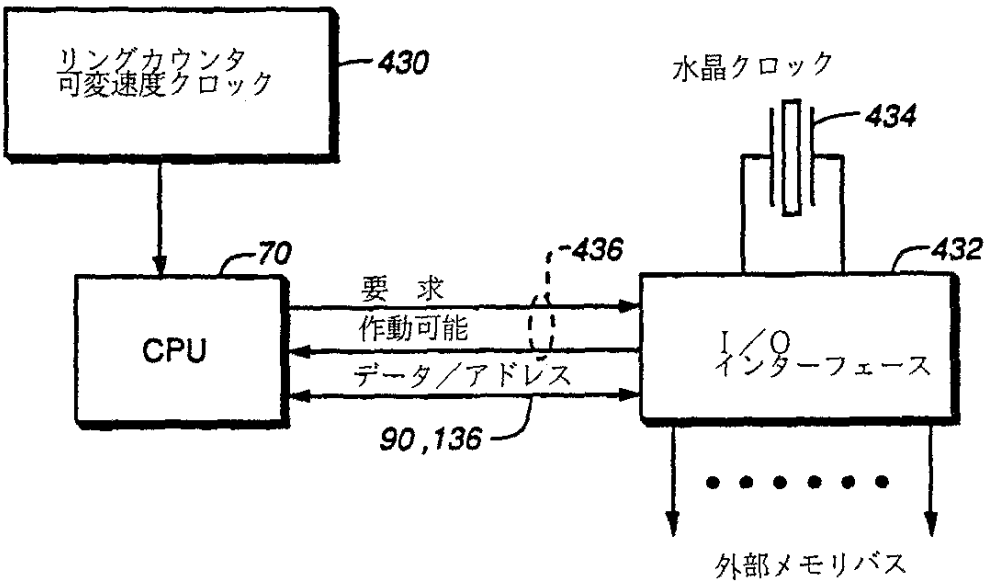
【第１２図】FIG. 12


CPU要求
(0) Rスタック
(1) スタック
(2) データ取り出し
(3) 命令取り出し
370
優先順位
DMA要求信号
CPU要求信号
内部アドレスバス
136
376
DMAカウンタ
378
MUX
優先順位
バス要求サービス
されるまで存在
バス要求
バス許可
372
バス制御
374
380
A23 — A13
A12 — A2
MUX
実行（メモリ
サイクルの発生）
メモリ作動可能
（アクセス完了）
402
400
CLE
384
リセット
内部データバス
A32 — A22
多重化アドレスバス
A32 — A13
A12 — A2
シフトレジスタ
190
390
392
388
394
396
386
MUX
MUX
OE
W
MUX
32
382
アドレス
／データ
RAS1
RAS2
CAS1
CAS2

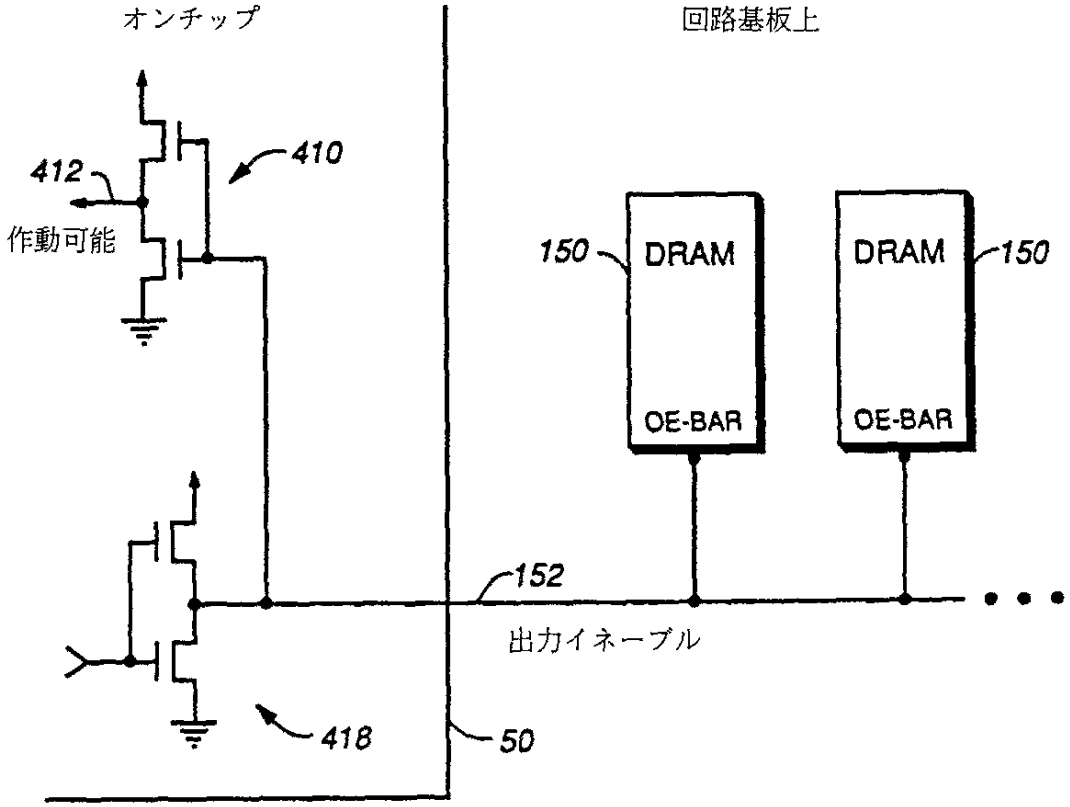
【第１３図】 FIG. 13



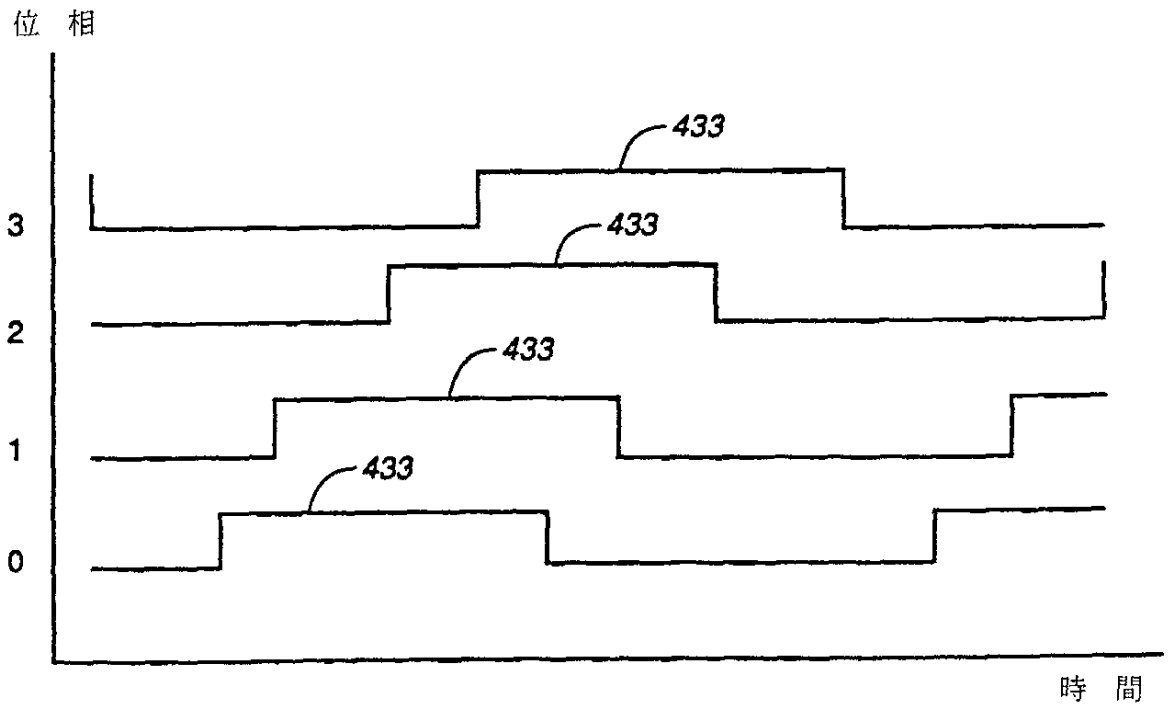
【第１７図】 FIG. 17



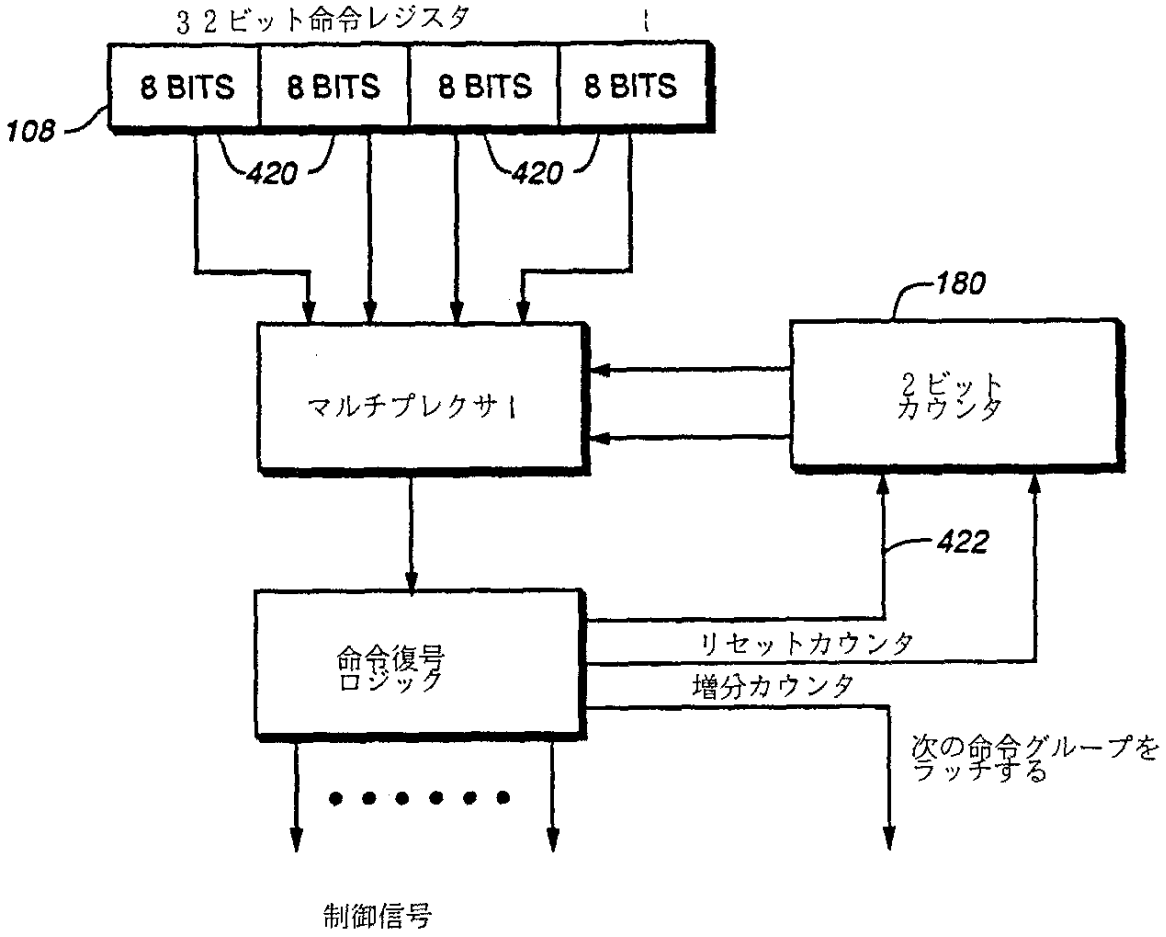
【第１４図】 FIG. 14



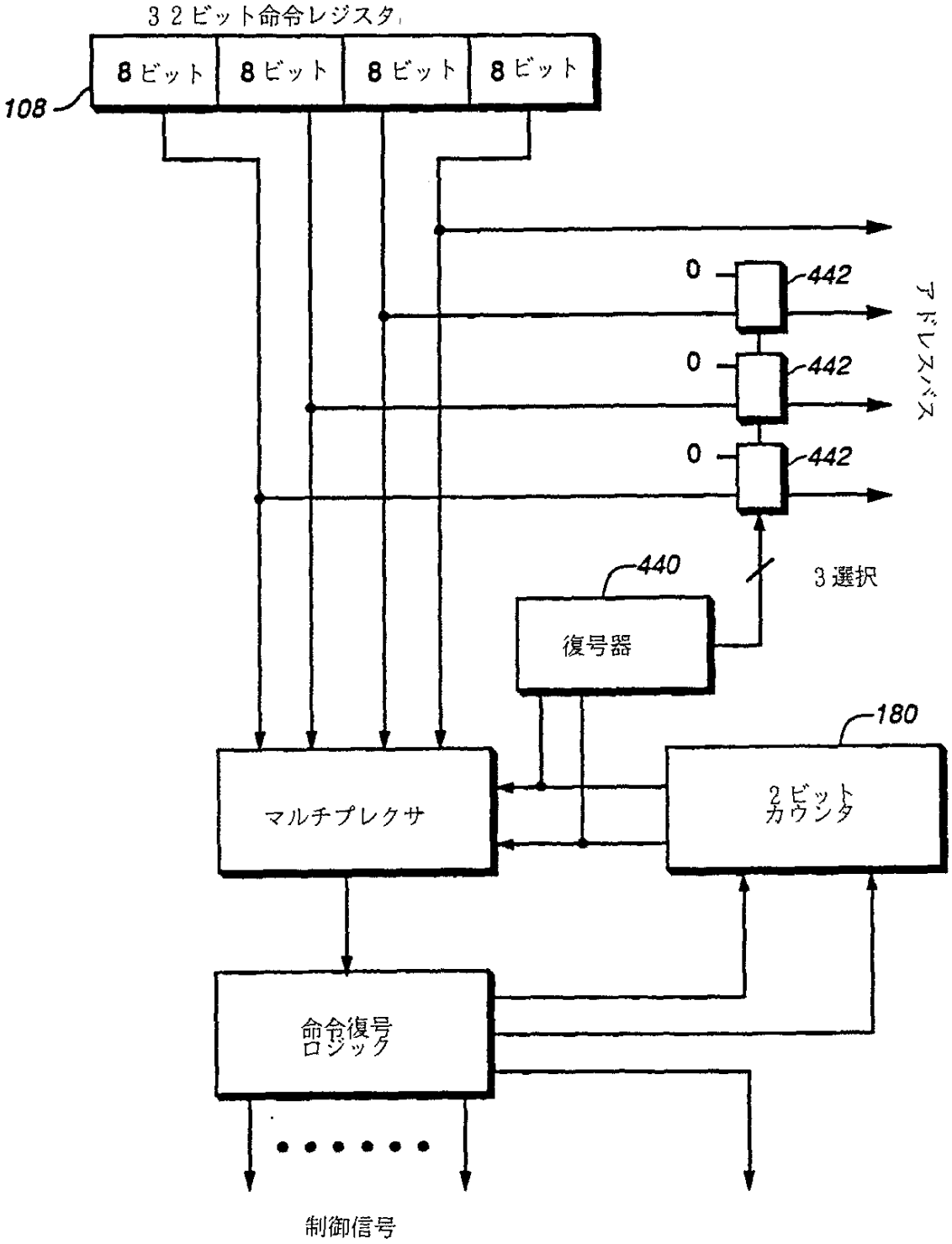
【第１９図】FIG. 19



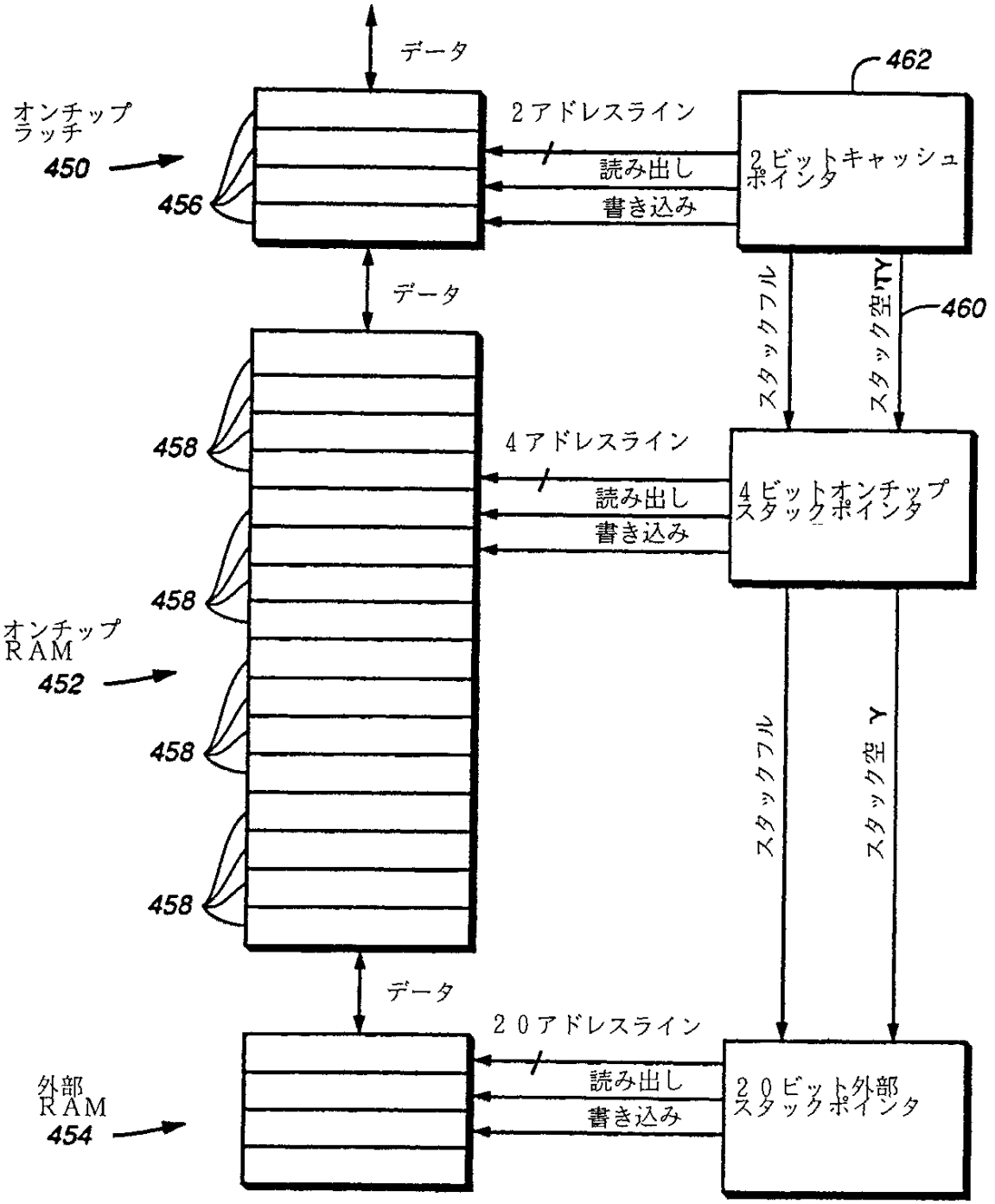
【第１６図】FIG. 16


３２ビット命令レジスタ
8 BITS
8 BITS
8 BITS
8 BITS
108
420
420
180
マルチプレクサ
２ビット
カウンタ
422
命令復号
ロジック
リセットカウンタ
増分カウンタ
次の命令グループを
ラッチする
制御信号

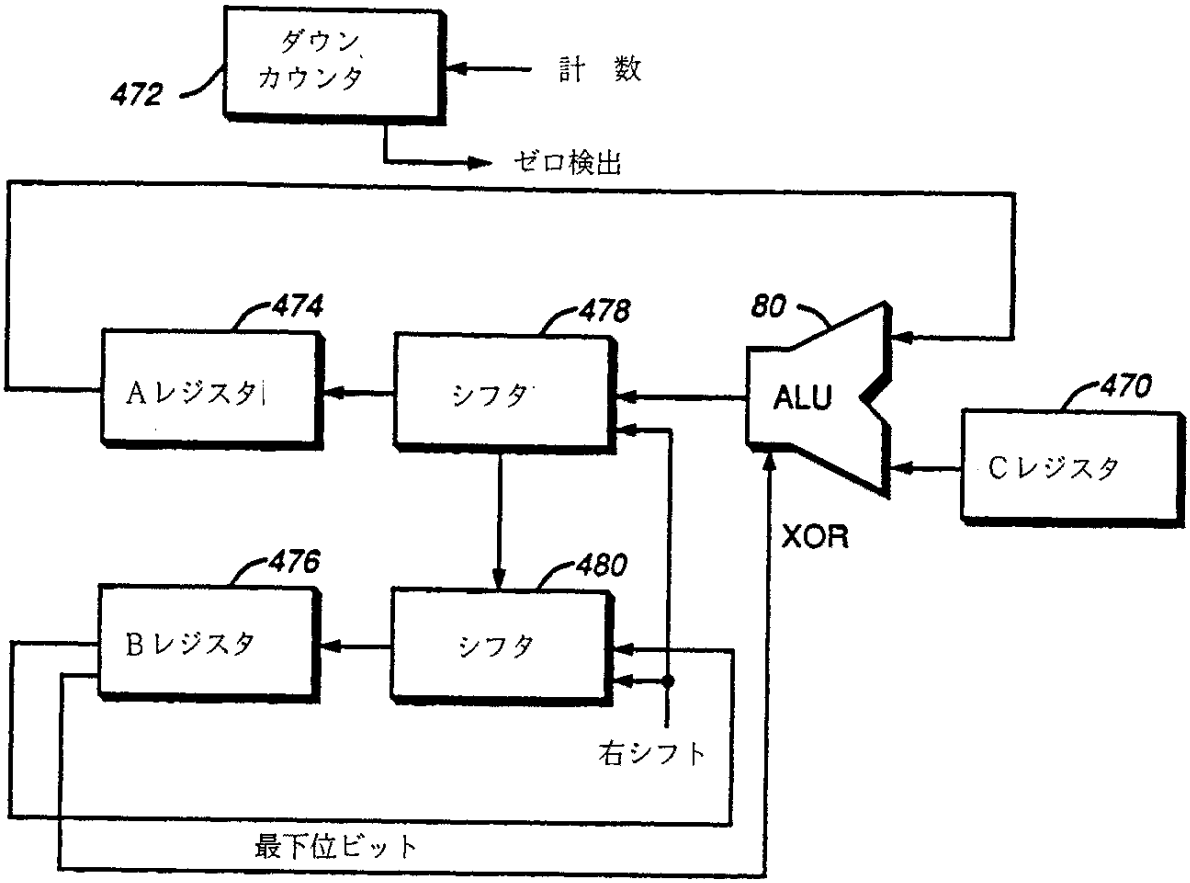
【第２０図】FIG. 20


３２ビット命令レジスタ
8ビット
8ビット
8ビット
8ビット
108
0
442
0
442
0
442
アドレスバス
440
復号器
３選択
180
マルチプレクサ
２ビット
カウンタ
命令復号
ロジック
制御信号

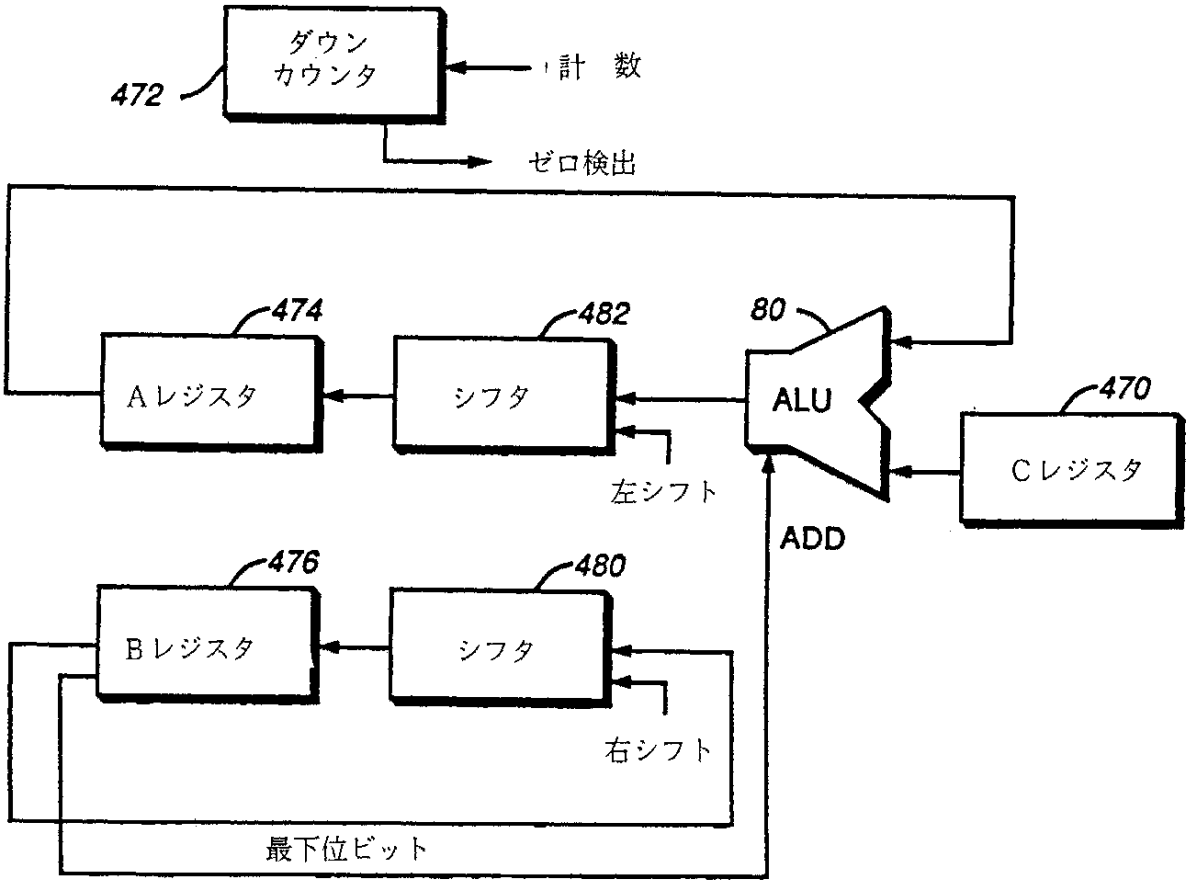
【第２１図】FIG. 21


データ
462
オンチップ
ラッチ
450
2アドレスライン
読み出し
書き込み
2ビットキャッシュ
ポインタ
456
データ
スタックフル
スタック空 Y
460
458
4アドレスライン
読み出し
書き込み
4ビットオンチップ
スタックポインタ
458
オンチップ
RAM
452
458
458
スタックフル
スタック空 Y
データ
20アドレスライン
読み出し
書き込み
20ビット外部
スタックポインタ
外部
RAM
454

【第２２図】 FIG. 22


ダウン
カウンタ
472
計 数
ゼロ検出
474
478
80
Aレジスタ
シフタ
ALU
470
Cレジスタ
XOR
476
480
Bレジスタ
シフタ
右シフト
最下位ビット

【第２３図】FIG. 23



フロントページの続き

(72)発明者　ムーア　チャールズ　エイチ
アメリカ合衆国　カリフォルニア州　94062　ウッドサイド　スター　ヒル　ロード　410

(72)発明者　フィッシュ　ラッセル　エイチ　ザ　サード
アメリカ合衆国　カリフォルニア州　94043　マウンテン　ヴィュー　スウィート　85　ショアライン　ブールヴァード　750

(56)参考文献　特開　昭57－20979（ＪＰ，Ａ）
特開　昭57－196334（ＪＰ，Ａ）
特開　昭63－200235（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.⁶，ＤＢ名)
G06F　9/38
G06F　9/34

Continuation from front page

(73) Patentee: Moore, Charles H.
410 Star Hill Road, Woodside, CA 94062
United States of America

(73) Patentee: Fish, Russell H., III
750 Shoreline Boulevard, Suite 85, Mountain View, CA 94062
United States of America

(56) Reference Literature: Publication of Unexamined Japanese Patent Application (*Kokai*) No. S57(1982)-20989 (JP, A)
Kokai No. S57(1982)-196334 (JP, A)
Kokai No. S63(1988)-200235 (JP, A)

(58) Fields Examined (Int. Cl.^{6}, DB name):
G06F 9/38
C06F 9/34